# Exercise: Functions

Problems for exercise and homework for the [Python Fundamentals Course @SoftUni](https://softuni.bg/trainings/3368/python-fundamentals-may-2021).   
Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/1728>.

## Smallest of Three Numbers

Write a function which receives **three integer** numbers and **returns** the **smallest**. Print the result on the console. Use appropriate name for the function.

def smallest\_func(a, b, c):  
 smallest\_num = 0  
 if a < b and a < c:  
 smallest\_num = a  
 elif b < a and b < c:  
 smallest\_num = b  
 elif c < a and c < b:  
 smallest\_num = c  
 return smallest\_num  
  
  
int\_a = int(input())  
int\_b = int(input())  
int\_c = int(input())  
  
print(smallest\_func(int\_a, int\_b, int\_c))

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  5  3 | 2 |
| 600  342  123 | 123 |
| 25  21  4 | 4 |

## Add and Subtract

You will receive **three** **integer numbers.**

Write functions named:

* sum\_numbers() which **returns** the **sum** of the **first two** integers
* subtract()which **returns** the **difference** between the **returned result** of the first function and the **third** integer.

Wrap the two functions in a function named **add\_and\_subtract()** which will receive the three numbers as parameters. Print the result of the subtract() function on the console.

def sdd\_and\_subtract(a, b, c):  
 result = (a + b) - c  
 return result  
  
  
int\_a = int(input())  
int\_b = int(input())  
int\_c = int(input())  
  
print(sdd\_and\_subtract(int\_a, int\_b, int\_c))

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 23  6  10 | 19 |
| 1  17  30 | -12 |
| 42  58  100 | 0 |

## Characters in Range

Write a function which receives **two characters** and **returns** a **single string** **with all the characters in** **between them** (according to the **ASCII** code), separated by a single **space**. Print the result on the console.

def char\_range\_func(a, b):  
 for i in range(a+1, b):  
 print(chr(i), end=' ')  
  
  
char\_1 = ord(input())  
char\_2 = ord(input())  
  
char\_range\_func(char\_1, char\_2)

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| a  d | b c |
| #  : | $ % & ' ( ) \* + , - . / 0 1 2 3 4 5 6 7 8 9 |
| #  C | $ % & ' ( ) \* + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B |

## Odd and Even Sum

You will receive a **single number.** You should write a function that returns the **sum** of **all even** and the **sum** of **all odd** digits in the given number as a single string in the format:

**"Odd sum = {sum\_of\_odd\_digits}, Even sum = {sum\_of\_even\_digits}"**

Print the result of the function on the console.

number\_input = input()  
  
  
def even\_odd\_func():  
 even = [int(x) for x in number\_input if int(x) % 2 == 0]  
 odd = [int(x) for x in number\_input if int(x) % 2 != 0]  
 return f"Odd sum = {sum(odd)}, Even sum = {sum(even)}"  
  
  
print(even\_odd\_func())

def odd\_even\_sum(a):  
 even\_list = []  
 odd\_list = []  
 for i in a:  
 i = int(i)  
 if i % 2 == 0:  
 even\_list.append(i)  
 else:  
 odd\_list.append(i)  
  
 print(f"Odd sum = {sum(odd\_list)}, Even sum = {sum(even\_list)}")  
  
  
input\_num\_str = str(input())  
  
odd\_even\_sum(input\_num\_str)

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1000435 | Odd sum = 9, Even sum = 4 |
| 3495892137259234 | Odd sum = 54, Even sum = 22 |

## Palindrome Integers

A palindrome is a number which reads the same **backward as forward**, such as 323 or 1001. Write a function which receives a **list of positive integers**, separated by comma and space **", "**. The function should **check** if **each integer** is a **palindrome** - True or False. Print the result.

def is\_palindrome\_func(a):  
 is\_palindrome = False  
  
 for each\_str in a:  
 if len(each\_str) == 1:  
 is\_palindrome = True  
 for i in range(0, int(len(each\_str) / 2)):  
 if each\_str[i] == each\_str[len(each\_str) - i - 1]:  
 is\_palindrome = True  
 else:  
 is\_palindrome = False  
  
 print(is\_palindrome)  
  
  
input\_list = input().split(', ')  
  
is\_palindrome\_func(input\_list)

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 123, 323, 421, 121 | False  True  False  True |  | 32, 2, 232, 1010 | False  True  True  False |

### Hints

* You could read more about palindromes here: <https://en.wikipedia.org/wiki/Palindrome>

## Password Validator

Write a function that checks if a given password is valid. Password validations are:

* It should be **6 - 10** (inclusive) characters long
* It should consist **only of** **letters** and **digits**
* It should have **at least 2** digits

If a password is **valid** print **"Password** **is** **valid"**.

Otherwise, for every unfulfilled rule print a message:

* **"Password must be between 6 and 10 characters"**
* **"Password must consist only of letters and digits"**
* **"Password must have at least 2 digits"**

def is\_password\_length\_valid(password: str):  
 if 6 <= len(password) <= 10:  
 return True  
 else:  
 return False  
  
  
def is\_password\_alphanumeric(password: str):  
 if password.isalnum():  
 return True  
 else:  
 return False  
  
  
def at\_least\_two\_digits(password: str):  
 digits\_count = 0  
 for ch in password:  
 if ch.isdigit():  
 digits\_count += 1  
  
 if digits\_count >= 2:  
 return True  
 else:  
 return False  
  
  
def validate\_password(password: str):  
 is\_valid = True  
  
 if not is\_password\_length\_valid(password):  
 print("Password must be between 6 and 10 characters")  
 is\_valid = False  
  
 if not is\_password\_alphanumeric(password):  
 print("Password must consist only of letters and digits")  
 is\_valid = False  
 if not at\_least\_two\_digits(password):  
 print("Password must have at least 2 digits")  
 is\_valid = False  
  
 if is\_valid:  
 print("Password is valid")  
  
password\_input = input()  
  
validate\_password(password\_input)

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| logIn | Password must be between 6 and 10 characters  Password must have at least 2 digits |
| MyPass123 | Password is valid |
| Pa$s$s | Password must consist only of letters and digits  Password must have at least 2 digits |

## Perfect Number

A perfect number is a **positive** integer that is equal to the **sum** ofits **proper positive divisors**. That is the sum of its positive **divisors** excluding the number itself (also known as its **aliquot sum**).

Write a function which receives an integer **number** and **returns** **one** of the following messages:

* **"We have a perfect number!"** - if the number is **perfect**.
* **"It's not so perfect."** - if the number is **NOT** **perfect**.

Print the result on the console.

number = int(input())  
  
  
def solve():  
 if number % 2 == 0:  
 s = 0  
 for i in range(1, number // 2 + 1):  
 if number % i == 0:  
 s += i  
 if s == number:  
 print('We have a perfect number!')  
 else:  
 print("It's not so perfect.")  
 else:  
 print("It's not so perfect.")  
  
  
solve()

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 6 | We have a perfect number! | 1 + 2 + 3 |
| 28 | We have a perfect number! | 1 + 2 + 4 + 7 + 14 |
| 1236498 | It's not so perfect. |  |

### Hint

Every perfect number is **half the sum** of all its positive divisors (including itself) => the sum of all positive divisors (all of which are divided without remainder) of 6 is 1 + 2 + 3 + 6 = 12. Half of 12 is 6 => 6 is perfect number.

* You could read more about the perfect number here: <https://en.wikipedia.org/wiki/Perfect_number>

## \* Loading Bar

You will receive a **single integer number** between **0** and **100** (inclusive) which is divisible by 10 without remainder (0, 10, 20, 30...). Your task is to create a function which returns a **loading bar** depending on the number you have received in the input. Print the result on the console. For more clarification see the examples below.

def loading\_bar(n):  
 ready = ("%"\*int(n/10))  
 remain = ("."\*int(10-n/10))  
 loading\_bar = ready+remain  
 return loading\_bar  
  
n = int(input())  
  
if n == 100:  
 print(f'100% Complete!\n[{loading\_bar(n)}]')  
else:  
 print(f'{n}% [{loading\_bar(n)}]\nStill loading...')

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 30 | 30% [%%%.......]  Still loading... |
| 50 | 50% [%%%%%.....]  Still loading... |
| 100 | 100% Complete!  [%%%%%%%%%%] |

## \* Factorial Division

Write a function that receives **two** integer numbers. Calculate **factorial** of each number. Divide the first result by the second and **print the division** formatted to the **second decimal** point.

from math import factorial  
  
number\_1 = int(input())  
number\_2 = int(input())  
  
factorial\_num\_1 = factorial(number\_1)  
factorial\_num\_2 = factorial(number\_2)  
  
result\_division = factorial\_num\_1 / factorial\_num\_2  
  
print(f"{result\_division:.2f}")

num\_1 = int(input())  
num\_2 = int(input())  
  
factorial\_num\_1 = 1  
factorial\_num\_2 = 1  
  
for i in range(1, num\_1 + 1):  
 factorial\_num\_1 \*= i  
  
for j in range(1, num\_2 + 1):  
 factorial\_num\_2 \*= j  
  
print(f"{factorial\_num\_1 / factorial\_num\_2:.2f}")

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 5  2 | 60.00 |  | 6  2 | 360.00 |

### Hints

* Read more about factorial here: <https://en.wikipedia.org/wiki/Factorial>

## \*List Manipulator

Trifon has finally become a junior developer and has received his first task. It is about manipulating a list of integers. He is not quite happy about it since he hates manipulating lists. They are going to pay him a lot of money, though, and he is willing to give somebody half of it if to help him do his job. You, on the other hand, love lists (and money) so you decide to try your luck.

The list may be manipulated by one of the following commands

* **exchange {index}** – splits the list **after** the given index and exchanges the places of the two resulting sub-lists. E.g. [1, 2, 3, 4, 5] -> exchange 2 -> result: [4, 5, 1, 2, 3]
  + If the index is outside the boundaries of the list, print "**Invalid index**"
  + Negative index is considered invalid.
* **max** **even/odd**– returns the **INDEX** of the max even/odd element -> [1, 4, 8, 2, 3] -> **max odd** -> print **4**
* **min** **even/odd** – returns the **INDEX** of the min even/odd element -> [1, 4, 8, 2, 3] -> **min even** > print **3**
  + If there are two or more equal **min/max** elements, return the index of the **rightmost** one
  + If a **min/max even/odd** element **cannot** be found, print **"No matches"**
* **first {count}** **even/odd**– returns the first {count} elements -> [1, 8, 2, 3] -> **first 2 even** -> print [**8, 2]**
* **last {count}** **even/odd** – returns the last {count} elements -> [1, 8, 2, 3] -> **last 2 odd** -> print [**1, 3]**
  + If the count is greater than the list length, print "**Invalid count**"
  + If there are **not** **enough** elements to satisfy the count, print as many as you can. If there are **zero** **even/odd** elements, print an empty list "[]"
* **end** – stop taking input and print the final state of the list

### Input

* The input data should be read from the console.
* On the first line, the initial list is received as a line of integers, separated by a single space
* On the next lines, until the command "**end**" is received, you will receive the list manipulation commands
* The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

* The output should be printed on the console.
* On a separate line, print the output of the corresponding command
* On the last line, print the final list in **square brackets** with its elements separated by a comma and a space
* See the examples below to get a better understanding of your task

### Constraints

* The **number of input lines** will be in the range [2 … 50].
* The **list elements** will be integers in the range [0 … 1000].
* The **number of elements** will be in the range [1 .. 50]
* The **split index** will be an integer in the range [-231 … 231 – 1]
* **first/last count** will be an integer in the range [1… 231 – 1]
* There will **not** be redundant whitespace anywhere in the input
* Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 3 5 7 9  exchange 1  max odd  min even  first 2 odd  last 2 even  exchange 3  end | 2  No matches  [5, 7]  []  [3, 5, 7, 9, 1] |
| **Input** | **Output** |
| 1 10 100 1000  max even  first 5 even  exchange 10  min odd  exchange 0  max even  min even  end | 3  Invalid count  Invalid index  0  2  0  [10, 100, 1000, 1] |
| **Input** | **Output** |
| 1 10 100 1000  exchange 3  first 2 odd  last 4 odd  end | [1]  [1]  [1, 10, 100, 1000] |

import sys  
  
  
def is\_valid\_index(collection: list, idx: int):  
 if 0 <= idx < len(collection):  
 return True  
 return False  
  
  
def exchange\_list\_at\_index(collection: list, idx: int):  
 exchanged\_list = []  
 if not is\_valid\_index(collection, idx):  
 print("Invalid index")  
 exchanged\_list = collection  
 else:  
 left\_sub\_list = collection[:idx + 1]  
 right\_sub\_list = collection[idx + 1:]  
  
 for i in right\_sub\_list:  
 exchanged\_list.append(i)  
 for i in left\_sub\_list:  
 exchanged\_list.append(i)  
  
 return exchanged\_list  
  
  
def max\_num(collection: list, custom\_filter):  
 # max\_element = -sys.maxsize  
 max\_element = float('-inf')  
 max\_element\_index = -1  
 # Returns -1 if there is no match.  
 # >= because the problem descriptions requires the "rightmost" element.  
 for i in range(len(collection)):  
 num = collection[i]  
 if custom\_filter(num) and num >= max\_element:  
 max\_element = num  
 max\_element\_index = i  
  
 if max\_element\_index == -1:  
 # No matches found.  
 print("No matches")  
  
 return max\_element\_index  
  
  
def min\_num(collection, custom\_filter):  
 min\_element = sys.maxsize  
 min\_element\_index = -1  
 # Returns -1 if there is no match.  
 # >= because the problem descriptions requires the "rightmost" element.  
 for i in range(len(collection)):  
 num = collection[i]  
 if custom\_filter(num) and num <= min\_element:  
 min\_element = num  
 min\_element\_index = i  
  
 if min\_element\_index == -1:  
 # No matches found.  
 print("No matches")  
  
 return min\_element\_index  
  
  
def first\_counted\_elements(collection: list, cnt: int, custom\_filter):  
 if cnt > len(collection):  
 print("Invalid count")  
 else:  
 matching\_elements = []  
 for num in collection:  
 if custom\_filter(num) and len(matching\_elements) < cnt:  
 matching\_elements.append(num)  
  
 print(matching\_elements)  
  
  
def last\_counted\_elements(collection: list, cnt: int, custom\_filter):  
 if cnt > len(collection):  
 print("Invalid count")  
 else:  
 matching\_elements = []  
 for i in range(len(collection)-1, -1, -1):  
 num = collection[i]  
 if custom\_filter(num) and len(matching\_elements) < cnt:  
 matching\_elements.append(num)  
 matching\_elements = matching\_elements[::-1]  
 # We have to reverse back the list,because we had already reversed it above.  
 print(matching\_elements)  
  
  
input\_numbers = input().split()  
numbers = [int(x) for x in input\_numbers]  
  
command = input()  
  
while command != 'end':  
 command\_arguments = command.split()  
 command\_type = command\_arguments[0]  
  
 if command\_type == 'exchange':  
 index = int(command\_arguments[1])  
 numbers = exchange\_list\_at\_index(numbers, index)  
  
 elif command\_type == 'max':  
 command\_filter = command\_arguments[1]  
 max\_index = -1  
 if command\_filter == 'even':  
 max\_index = max\_num(numbers, lambda n: n % 2 == 0)  
 elif command\_filter == 'odd':  
 max\_index = max\_num(numbers, lambda n: n % 2 != 0)  
 if max\_index != -1:  
 print(max\_index)  
  
 elif command\_type == 'min':  
 command\_filter = command\_arguments[1]  
 min\_index = -1  
 if command\_filter == 'even':  
 min\_index = min\_num(numbers, lambda n: n % 2 == 0)  
 elif command\_filter == 'odd':  
 min\_index = min\_num(numbers, lambda n: n % 2 != 0)  
 if min\_index != -1:  
 print(min\_index)  
  
 elif command\_type == 'first':  
 count = int(command\_arguments[1])  
 command\_filter = command\_arguments[2]  
 if command\_filter == 'even':  
 first\_counted\_elements(numbers, count, lambda n: n % 2 == 0)  
 elif command\_filter == 'odd':  
 first\_counted\_elements(numbers, count, lambda n: n % 2 != 0)  
  
 elif command\_type == 'last':  
 count = int(command\_arguments[1])  
 command\_filter = command\_arguments[2]  
 if command\_filter == 'even':  
 last\_counted\_elements(numbers, count, lambda n: n % 2 == 0)  
 elif command\_filter == 'odd':  
 last\_counted\_elements(numbers, count, lambda n: n % 2 != 0)  
  
 command = input()  
  
print(numbers)