[1. A 2](#_Toc147443225)

[2. Booleans 4](#_Toc147443226)

[3. Comprehensions 5](#_Toc147443227)

[4. Debugger 6](#_Toc147443228)

[5. Decimal 7](#_Toc147443229)

[6. Dictionaries 7](#_Toc147443230)

[7. Error-Handling 11](#_Toc147443231)

[8. File Handling 13](#_Toc147443232)

[9. Formatting, Printing 15](#_Toc147443233)

[10. Functions 16](#_Toc147443234)

[11. Imports 16](#_Toc147443235)

[12. Lists 17](#_Toc147443236)

[13. Matrix 18](#_Toc147443237)

[14. OOP 18](#_Toc147443238)

[14.1. First-Steps-in-OOP 19](#_Toc147443239)

[14.2. Classes-and-Objects 19](#_Toc147443240)

[14.3. Inheritance 19](#_Toc147443241)

[15. PyCharm 20](#_Toc147443242)

[15.1. Shortcuts 20](#_Toc147443243)

[15.2. Settings 24](#_Toc147443244)

[16. Referenced 25](#_Toc147443245)

[17. Regex 26](#_Toc147443246)

[18. Sets - кортежи(tuple) и множества(set) 29](#_Toc147443247)

[19. Shortcuts 30](#_Toc147443248)

[20. Slicing 31](#_Toc147443249)

[21. Symbol names 32](#_Toc147443250)

[22. Text 33](#_Toc147443251)

[23. Time 33](#_Toc147443252)

[24. Tuples - кортежи(tuple) и множества(set) 35](#_Toc147443253)

[25. ZZZ Other 36](#_Toc147443254)

[25.1. If… Else … replacement 36](#_Toc147443255)

# A

self.is\_on = not self.is\_on

# change “on” to “off” or vice versa

Referenced list, ????

biggest\_sum = -float("inf")

Dunder – double underscore ???

**round\_half\_correctly.py**

snake**\_**case

**P**ascal**C**ase

camel**C**ase

Mangling

**or** has a lower priority than **and**

**and** has a lower priority than **not**

Parameters

Arguments

Attributes

print(f"Milk: {', '.join(str(x) for x in cups) or 'empty'}")

@staticmethod  
def find\_object(collection: list, attribute: str, value: str):  
 for obj in collection:  
 if str(getattr(obj, attribute)) == value:  
 return obj

print(isinstance('a', int)) # False  
print(isinstance(5, int)) # True

from functools import reduce

map\_functions = {

'\*': lambda x: reduce(lambda a, b: a \* b, x),

'/': lambda x: reduce(lambda a, b: a / b, x),

# '/': lambda x: reduce(lambda a, b: a + b if a == 0 or b == 0 else a / b, x),

'+': lambda x: reduce(lambda a, b: a + b, x),

'-': lambda x: reduce(lambda a, b: a - b, x),

} 02\_expression\_evaluator\_a.py in 03\_Stacks\_Queues\_Tuples\_and\_Sets\_Exercise

summation\_pairs.py

W:\1\_Python\1-Training\1\_Projects\1st\_Project\03\_Advanced

\02\_Tuples\_and\_Sets\Lab\6\_summation\_pairs.py

command = "Replace-{file\_name}-{old\_string}-{new\_string}"  
action, \*info, last = command.split('-')

a, b, c = 2, '\*', 3  
print(eval(f"{a}{b}{c}")) # 6

eval is slow and info inside eval could be stolen from hacker

if ("Doll" and "Wooden") in crafted:# Wrong!!!

if "Doll" in crafted and "Wooden" in crafted**:** # Correct!!!

for i in range(0, 2, **0.5**):  
 print(i)

**TypeError: 'float' object cannot be interpreted as an integer**

for i in range(0, 5, **int**(0.5)):  
 print(i, end=' ')

**ValueError: range() arg 3** (int(0.5)) **must not be zero**

Python is a **dynamic** language

Variables are **not** directly associated with

any particular value type

Any variable can be **assigned** (and **re-assigned**)

values of all types

x = 2.45 # float   
y = 5 # int  
w = x // 2 # float - take the class of x  
print(type(w)) # class 'float'

w = y // 2 # int - changing to the class of y  
print(type(w)) # class 'int'

Python **integers** are **immutable**

Python **floats** are **immutable**

Python **strings** are **immutable**

This means that once a string is created,

it is **not** possible to **modify** it

name = 'George'  
name[0] = 'P' # Error не може да променим G

print(name) # George

name = 'Ime' # заделя друго място в паметта

различно от мястото за George  
print(name) # Ime  
name = 4 # заделя трето място в паметта  
print(name) # 4

string **interpolation** are string **literals**(буквален)

that allow **embedded**(вградени) expressions

result = first\_number **//** second\_number # **integer division**  
result = first\_number **%** second\_number # **modular division**

result = first\_number **/** second\_number # result is **always float**

“Prime number” Просто число

“Complex number”

# *TODO: Add logic here*# *TODO: Check the other cases…*

# Booleans

self.is\_on = not self.is\_on

# change “on” to “off” or vice versa

print(bool(0)) # False  
print(bool(-0)) # False  
print(bool("")) # False  
print(bool(" ")) # True  
print(bool(False)) # False  
print(bool(None)) # False  
print(bool(True)) # True  
print(bool(1)) # True  
print(bool("a")) # True

# Comprehensions

action, \*info, last = command.split('-')

action, \*indexes = [int(x) if x.isdigit() else x for x in command.split()]

return {r1, c1}.issubset(range(rows)) and {r2, c2}.issubset(range(cols))

set1 = {input() for \_ in range(n)}

data1, data2 = [list(map(int, el.split(','))) for el in input().split('-')]

materials.reverse() <=> materials[:: -1]

[print(f"{toy}: {crafted.count(toy)}") for toy in sorted(set(crafted))]

matrix = [[int(x) for x in input().split(", ")] for \_ in range(int(input().split(", ")[0]))]

# Debugger

<https://softuni.bg/trainings/resources/video/86023/video-28-june-2023-ines-kenova-python-oop-june-2023/4108> - 11 minute

 right click over the existing break point (brake) stops if command = = “apple” - this is new statement different from the one in file

# Decimal

from decimal import localcontext, Decimal, ROUND\_HALF\_UP, ROUND\_HALF\_DOWN

**round\_half\_correctly.py**

import **decimal** - in **decimal.py**

**ERROR скапах всичко**

a = Decimal('0.1')  
b = Decimal('0.1')  
c = Decimal("0.1")

result = a + b + c # 0.3

a = 0.1  
b = 0.1  
c = 0.1

result = a + b + c # 0.30000000000000004

a = Decimal(0.1) # without apostrophe  
b = Decimal(0.1) # without apostrophe  
c = Decimal(0.1) # without apostrophe  
result = a + b + c # 0.3000000000000000166533453694  
price = Decimal("3 \* 1.2") Error

price = Decimal("3 + 1.2") Error

no operations allowed, just one number

# Dictionaries

symbols[ch] = symbols.get(ch, 0) + 1

dict\_test = {3: 4, 4: 5, 5: 5, 7: 2, 11: 2}  
print(len(dict\_test))  
# dict\_test1 = {"k3": 4, "k4": 5, "k5": 5, "k7": 2}  
# sorted\_dict = dict(sorted(dict\_test.items(), key=lambda x: (x[1], x[0])))  
# print(sorted\_dict)  
# sorted\_dict = dict(sorted(dict\_test1.items(), key=lambda x: (x[1], x[0])))  
# print(sorted\_dict)  
# race\_info = sorted(race\_info, key=lambda x: -race\_info[x]) # returns list with keys sorted by values  
  
# sorted(symbols.items()) # returns list of tuples  
# dict\_test = dict(sorted(symbols.items()))  
# for ch, count in dict\_test.items():  
# print(f"{ch}: {count} time/s")  
# for ch, count in sorted(dict\_test.items()):  
# print(f"{ch}: {count} time/s")

from collections import defaultdict  
# from collections import OrderedDict  
  
# student\_info = defaultdict(list)  
# # student\_info = defaultdict(lambda: [0.0])  
# for \_ in range(int(input())):  
# name, grade = input().split()  
# # if name not in student\_info: this check can be omitted with defaultdict  
# # student\_info[name] = []  
# student\_info[name].append(float(grade))  
  
# x = ('key1', 'key2', 'key3')  
# y = 0, 1, 2  
# this\_dict = dict.fromkeys(x)  
# # this\_dict = dict.fromkeys(x, y)  
# print(this\_dict)  
# this\_dict = dict(zip(x, y))  
# print(this\_dict)  
#  
# txt = "Hello, welcome to my world."  
# print(txt.find("q")) # -1 or index if q in txt  
# print(txt.index("q")) # Error or index if q in txt  
#  
car = {  
 "brand": "Ford",  
 "model": "Mustang",  
 "year": 1964  
}  
# x = car.items()  
# print(car)  
# print(type(car))  
# print(x)  
# print(type(x))  
# for key, value in car.items():  
# print(key, value)  
  
# x = car.setdefault("model", "Bronco") # return Mustang if key exists  
# print(x)  
# print(car)  
# y = car.setdefault("mod", "Bronco") # add it and return Bronco if key does not exist  
# print(y)  
# print(car)  
#  
# car.update({"model": "laguna"}) # change value if key exists  
# print(car)  
# car.update({"test": "New\_mod"}) # add key, value if key does not exist  
# print(car)  
# car["li"] = 5 # act as update  
# print(car)  
# car["model"] = "lag" # act as update  
# print(car)  
  
# bus = {  
# "br": "Fo",  
# "model": "Mus",  
# "ye": 19  
# }  
# # car.setdefault(bus) # Error - requires (key, value)  
# # car.update("model", "Bronco") # Error - requires dict  
# car.update(bus) # requires dict {key, value}  
# print(car)  
  
  
# x = car.get("br", ) # None  
# print(x)  
# x = car.get("br", 47) # 47  
# print(x)  
# y = car["br"] # Error  
# print(y)  
  
# x = car.keys() # Returns a list containing the dictionary's keys  
# x = car.values() # Returns a list of all the values in the dictionary  
  
# for el in car.items(): # !!!! tuple is the answer  
# print(el)  
  
# car.popitem() # Removes the last inserted key-value pair  
# car.pop("br") # Removes key-value pair or Error  
# car.pop("br", defaultvalue) returns defaultvalue and no Error  
  
#  
# a = ("a", "b", "c", "d")  
# a = ("a", "b")  
# b = ("1", "2", "3")  
# x = zip(a, b)  
# # print(tuple(x))  
# print(x)  
# print(dict(x))  
  
  
# print({ch: ord(ch) for ch in input().split(',')})  
  
# data = [("Peter", 22), ("Amy", 18), ("George", 35)]  
# dict\_data = {key: value for (key, value) in data}  
# print(dict\_data)  
# print(f"{key}: {value} for (key, value) in data}") # do not work  
  
# x = "012"  
# y = "01234567"  
# for i in range(len(y)):  
# j = i % len(x)  
# print(i, j, sep='->')  
  
# print(list(car.items()))  
# print(car['model'])

# Error-Handling

methods are faster than try except!!!  
Syntax errors(parsing errors) Exceptions

times = "asd"  
print(7 / times) # TypeError: unsupported operand type(s) for /: 'int' and 'str'  
print("7" / times) # TypeError: unsupported operand type(s) for /: 'str' and 'str'  
print(7 / int(times)) # ValueError: invalid literal for int() with base 10: 'asd'  
print(int("asd")) # ValueError: invalid literal for int() with base 10: 'asd'  
print(int([11])) # TypeError: int() argument must be a string, a bytes-like object or a real number, not 'list'

try:  
 times = int(input())  
 # times = float(input())  
except ValueError as ex:  
 print(f"ValueError: {ex}")

print("blabla")  
except KeyError:  
 print()  
except (NameError, TypeError, IndexError) as ex:  
 print(ex)

# custom exceptions  
class SmallValueException(Exception):  
 pass  
  
  
class HighValueException(Exception):  
 pass  
  
  
amount = float(input()) # you cannot transfer negative money  
  
if amount < 1:  
 raise SmallValueException("Amount can not be less than 1lv.")  
elif amount > 1000:  
 raise HighValueException("Transaction limit max 1000")  
# custom exceptions

try:  
 print("try")  
 a = 7  
 b = int(input()) # if b = 0 print("End") would not be executed, but print("finally")  
 c = a / b  
except ValueError as text:  
 print("ValueError") # ValueError  
 print(text) # invalid literal for int() with base 10: 'dhhfd'  
else:  
 print("from else") # Not very useful. will be executed if successful try.  
finally:  
 print("finally") # will always be executed  
  
print("End") # if b = 0, code could not reach that line, because of error. if b = 'str' will print End.  
# if b = 0 -> ZeroDivisionError. if b = 'str' ValueError.

# File Handling

**io** (in / out) module is the default module for accessing files - Built-in

file = open('W:/1\_Python/1-Training/1\_Projects/1st\_Project/text.py') correct  
file = open('W:\1\_Python\1-Training\1\_Projects\1st\_Project\text.py') **wrong**

We should always make sure that an open file is properly **closed**

To avoid **unwanted** **behaviour** **always** **close** the files

Files opened with “**with”** statement will be **closed** **automatically** once it leaves the **with** block

with open("file.txt", "w") as f:  
 f.write("Hello World!!!")

print(f.read()) # Error: io.UnsupportedOperation:

f is not readable if the file is open for writing, adding …

modes 'w', 'a' ….etc

* + **w** - open for writing, truncating the file first. Truncating(съкращавам) - If the file exists, its **overwritten**
  + **x** - create a new file and open it for writing
  + **r** – open in reading mode. ‘r’ is by default. No diff, If ‘r’ or mode is empty.
  + **a** - open for writing, appending to the end of the file. Or create a file, if it doesn’t exists.
  + **t** - text mode (default)
  + **b** - binary mode
  + **+** - open a disk file for updating (reading and writing)
* try:  
   file = open('zzz\_text.py', 'r')  
   print(file.read())  
  except FileNotFoundError:  
   print("File not found or path is incorrect")  
  finally:  
   print("exit")

file = open('text.txt') # => open('python.txt', 'r')  
print(file.read())  
print(file.read(7)) # will print nothing if file has been read already  
print(file.readline())  
print(file.readline(7))  
for line in file: # line is str + \n  
 # print(line) # adds additional empty line after printing each line of file  
 print(line, end="") # will print nothing if file has been read already  
 print(line.split())  
print(file.read()) # will print nothing if file has been read in

"for line in file" already  
file.close()

Delete File  
import os  
  
file\_path = "text.txt"  
if os.path.exists(file\_path):  
 os.remove(file\_path)  
  
try:  
 os.remove('text.txt')  
except FileNotFoundError:  
 print('File already deleted!')

# region Directory manipulation  
import os

os.path.isfile(path) # method that returns True if the path is a file or a symlink(symbolic link) to a file.  
os.path.exists(path) # method that returns True if the path is a file, directory, or a symlink(symbolic link) to a file.

# print(os.mkdir('W:/1\_Python/1-Training/1\_Projects/1st\_Project/Lessons\_Notes/File\_Handling\_Notes/Test\_Folder'))  
print(os.getcwd()) # Return a string representing the current working directory.  
# os.mkdir('Test')  
# os.rmdir('W:/1\_Python/1-Training/1\_Projects/1st\_Project/Lessons\_Notes/File\_Handling\_Notes/Test\_Folder')  
# os.chdir('Test\_Folder')  
print(os.listdir('W:/1\_Python/1-Training/1\_Projects/1st\_Project'))  
  
# endregion

# Formatting, Printing

int(5 / 2) ⬄ 5 // 2

orders = list("abcdef")

print("Orders left: ", end='')  
print(\*orders, sep=', ') # \* splat operator

print("Orders left:", \*orders, "text", '.')

print(int(1.5)) # 1

print(f"{minutes}:{seconds:02d}") # 5:07

print(f"{num:.1f}") # 1 -> 1.0 ; 1.333 -> 1.3

print(round(**4.5**)) #-> **4** round to nearest even number  
print(round(**5.5**)) #-> **6** banker's number

x = 4.5  
print(f'{x:.0f}') #-> **4** round to nearest even number  
x = 5.5  
print(f'{x:.0f}') #-> **6** banker's number



# Functions

def add\_number\_12(num\_seq):  
 num\_seq.append(12)  
# no return, but list nums is modified

nums = [1, 2, 3]  
print(nums) # [1, 2, 3]  
add\_number\_12(nums) # no return, but list nums is modified, because lists are referenced. num\_seq and nums are pointing to one and the same place in memory  
print(nums) # [1, 2, 3, 12]

# Imports

from string import punctuation # !"#$%&'()\*+,-./:;<=>?@[\]^\_`{|}~

import math  
x = 5.98  
print(math.floor(x)) -> not floor(x)  
print(int(x)) => floor(x)

from math import ceil, floor  
x = 5.98  
print(floor(x)) -> not math.floor(x)

import **decimal** - in **decimal.py**

**ERROR скапах всичко**

import random  
number = random.randint(1, 100)  
print(number)

from functools import reduce

map\_functions = {

'\*': lambda x: reduce(lambda a, b: a \* b, x),

'/': lambda x: reduce(lambda a, b: a / b, x),

# '/': lambda x: reduce(lambda a, b: a + b if a == 0 or b == 0 else a / b, x),

'+': lambda x: reduce(lambda a, b: a + b, x),

'-': lambda x: reduce(lambda a, b: a - b, x),

} 02\_expression\_evaluator\_a.py in 03\_Stacks\_Queues\_Tuples\_and\_Sets\_Exercise

from string import ascii\_lowercase  
chars = list(ascii\_lowercase)

py -m pip install PyQt5

py -m pip install pyfiglet or keep the cursor over the library and click install

py -m pip install opencv-python

# Lists

Be very careful with **remove** in **for** cycle!!!

nums = [1, 2, 3]  
nums2 = nums # referenced  
nums3 = nums.copy() ⬄ list(nums) # not referenced

bottles = list(map(int, input().split()))

my\_list = list(range(5))) # [0, 1, 2, 3, 4]

enumerate <class 'enumerate'>

print(list(enumerate(list("123")))) # [(0, '1'), (1, '2'), (2, '3')]

print(list(enumerate(list(range(3))))) # [(0, 0), (1, 1), (2, 2)]

indexes = [idx for idx, el in enumerate(test\_tuple) if el == “asd”]

return list with idx for all el == “asd”

x = [[]] \* 3 # [[], [], []]  
x[1].append(5) # [[5], [5], [5]] !!!  
a = [0] \* 3 # [0, 0, 0]  
a[2] += 7  
print(a) # [0, 0, 7]

my\_list = [1, 2, 3, 1, 2, 2, 2, 2, 4, 5, 'a']  
result = list(filter(lambda x: x == 2, my\_list)) # [2, 2, 2, 2, 2]  
result1 = next(filter(lambda x: x == 2, my\_list)) # 2  
result2 = next(filter(lambda x: x == 7, my\_list), "Not in list") # Not in list  
# result3 = next(filter(lambda x: x == 7, my\_list)) # StopIteration (error)

a = [1, 2, 3]  
b = ['w', 'f']  
d = [\*a, \*b]  
print(d) # [1, 2, 3 'w', 'f']  
print(\*d) # 1 2 3 w f

a = "12345"  
b = list(a) # ['1', '2', '3', '4', '5']

# removing elements in the middle of the list

a\_nums = a\_nums[:left\_idx] + a\_nums[right\_idx + 1:]  
print(a\_nums)  
# =>  
for i in range(idx + value, idx - value - 1, -1):  
 b\_nums.pop(i)  
print(b\_nums)  
# =>  
del c\_nums[left\_idx:right\_idx + 1]  
print(c\_nums)

# Matrix

matrix = [[0 for j in range(2)] for i in range(3)] # [[0, 0], [0, 0], [0, 0]]  
matrix = [[0 for \_ in range(2)] for \_ in range(3)] # [[0, 0], [0, 0], [0, 0]]

matrix = [[int(j) for j in input().split(", ") if int(x) % 2 == 0] for i in range(int(input()))]

matrix = [[int(x) for x in input().split(", ") if int(x) % 2 == 0] for \_ in range(int(input()))]

# flattening matrix 2d

matrix = [[int(j) for j in input().split(", ")] for i in range(int(input()))]  
flatten\_matrix = [el for list\_i in matrix for el in list\_i]

# flattening matrix 3d  
m3d = [[[k for k in range(3)] for j in range(3)] for i in range(3)]  
print(m3d) # [[[0, 1, 2], [0, 1, 2], [0, 1, 2]], [[0, 1, 2], [0, 1, 2], [0, 1, 2]], [[0, 1, 2], [0, 1, 2], [0, 1, 2]]]  
flatten\_m3d = [k for m2d in m3d for list\_i in m2d for k in list\_i]  
print(flatten\_m3d) # [0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2]

# sum of primary or secondary diagonal  
n = int(input())  
matrix = [[int(x) for x in input().split()] for \_ in range(n)]  
primary\_diagonal = sum([matrix[i][i] for i in range(n)])  
secondary\_diagonal = sum([matrix[i][n - i - 1] for i in range(n)])  
print(primary\_diagonal)  
print(secondary\_diagonal)  
  
# faster solution  
primary\_diagonal\_sum = 0  
secondary\_diagonal\_sum = 0  
for i in range(n):  
 row = [int(x) for x in input().split()]  
 primary\_diagonal\_sum += row[i]  
 secondary\_diagonal\_sum += row[n - i - 1]  
print(primary\_diagonal\_sum)  
print(secondary\_diagonal\_sum)

# OOP

from sys import path  
print(\*path, sep="\n") # prints Source Root Directories

Mangling

## First-Steps-in-OOP

Ssffhsfghsh

Hhdss

## Classes-and-Objects

Mangling

## Inheritance

\s\addaa

# PyCharm

## Shortcuts





Successively press Alt+J to find and select the next occurrence of case-sensitively matching word or text range. To remove selection from the last selected occurrence, press Alt+Shift+J

After the second or any consecutive selection was added with Alt+J, you can skip it and select the next occurrence with F3. To return the selection to the lastly skipped occurrence, press Shift+F3

Press Ctrl+Alt+Shift+J to select all case-sensitively matching words or text ranges in the document.

To redo Ctrl + Shift + Z

Ctrl+Alt+T - To surround with (if or try or ….)

To select multiple fragments (create multiple cursors) in the press and hold **Ctrl+Alt+Shift** and drag the mouse (Windows and Linux):

Press   Alt   F7   to quickly locate all occurrences of code referencing the symbol at the caret, no matter if the symbol is a part of a class, method, field, parameter, or another statement.

To toggle between the upper and lower case for the selected code fragment, press Ctrl+Shift+U



**Ctrl + Enter** new raw while caret stays

Complete statement **Shift + Enter** (Ctrl + Shift + Enter)

Start new line with - **Ctrl + Shift + Enter** (Shift + Enter)

**Ctrl + Alt + L** **or Ctrl+ ** automatically format code with spaces and lines

Move Caret To Code Block End with - **Ctrl + right bracket ]**

[Extend selection](https://www.jetbrains.com/help/pycharm/working-with-source-code.html) - **Ctrl+W**

Decrease selection - **Ctrl+Shift+W or Ctrl+** 

Select Several Rows To Be Simultaneously Edited - **Mouse Middle Click**

Duplicate current line or selection - **Ctrl + D**

Comment with line comment - **Ctrl + /**

New Python File - **Shift + Right Mouse Click**





To scroll a file horizontally, **turn the** **mouse wheel** while keeping **shift** pressed

Press **Ctrl + Shift + V** to select the text fragment that you have previously copied to the clipboard

Press **Ctrl + Shift + mouse**  to select the text word by word fragm

Press **Ctrl + `** - **zoomit** command

Mouse Middle Click or **Alt + shift + left mouse click** - select several rows to be simultaneously edited

Move Caret To Code Block End - **Ctrl +]**















## Settings



# Referenced

**List, Set, Dictionary – mutable – referenced** – it’s pointing to place in memory, even if you change it. But if you reassigned it would point to a different place in memory.

**Int, str, float, tuple, frozenset – immutable – not referenced.**

If you change it, it’ll point different place in memory

# All values in Python are references. What you need to worry about is if a type is mutable. The basic numeric and string types, as well as tuple and frozenset are immutable; names that are bound to an object of one of those types can only be rebound, not mutated.

a = 10  
b = a   
a = 30 # now a = 30 but b remains 10

list1 = [10,20,30,40]  
list2 = list1 #[10,20,30,40] list1 and list2 are one and the same object  
list1 = [3,4] # this list1 is different from the list1 up, because it’s reassigned ( it’s different object, written on a different place in memory and it’s not possible to invoke list1 anymore)  
# list1 ==> [3,4]  
# list2 ==> [10,20,30,40]

-------------------------------------------------------------------------------------

list1 = [10,20,30,40]  
list2 = list1 #[10,20,30,40] - one and the same object  
# change value of list 1 at a certain index say index 0  
list1[0] = 500 # now list1 is the same object as list1 with changed attribute value – mutated value  
# If you check again the values of list1 and list2 you will be surprised.  
#list1 ==> [500,20,30,40]  
#list2 ==> [500,20,30,40]

--------------------------------------------------------------------------------------

Set

a = {"a", "b", "c"}  
b = a  
a.add("d")  
print(a) # {'d', 'a', 'c', 'b'}  
print(b) # {'d', 'a', 'c', 'b'} set b is also changed

--------------------------------------------------------------------------------------

Dictionary

a = {"a": 1, "b": 2, "c": 3}  
b = a  
a["a"] = 7  
print(a) # {'a': 7, 'b': 2, 'c': 3}  
print(b) # {'a': 7, 'b': 2, 'c': 3} dictionary b is also changed

------------------------------------------------------------

def add\_number\_12(num\_seq):  
 num\_seq.append(12)  
# no return, but list nums is modified

nums = [1, 2, 3]  
print(nums) # [1, 2, 3]  
add\_number\_12(nums) # no return, but list nums is modified, because lists are referenced. num\_seq and nums are pointing to one and the same place in memory  
print(nums) # [1, 2, 3, 12]

-------------------------------------------------------------

def update\_set(num\_seq):  
 num\_seq.update("a", "s")  
# no return, but set nums is modified  
nums = {1, 2, 3}  
print(nums) # {1, 2, 3}  
update\_set(nums) # no return, but set nums is modified, because sets are referenced. num\_seq and nums are pointing to one and the same place in memory  
print(nums) # {1, 2, 3, 's', 'a'}

-----------------------------------------------------------

def update\_dictionary(num\_seq):  
 num\_seq.update({7: "s"})  
# no return, but dictionary nums is modified  
nums = {1: "z", 2: "x", 3: "e"}  
print(nums) # {1: 'z', 2: 'x', 3: 'e'}  
update\_dictionary(nums) # no return, but dictionaries nums is modified, because dictionaries are referenced. num\_seq and nums are pointing to one and the same place in memory  
print(nums) # {1: 'z', 2: 'x', 3: 'e', 7: 's'}

# Regex

import re  
  
([0]|[1-9][0-9]\*) -> matches 0 but not 00 or 01  
(?: ) - does not capture/assign a group ID.  
( ) - group with ID. \+359([\s-])\d\1 -> \1 recall group with ID=1 ([\s-])  
(?P<name> ) - group with name. \+359(?P<sep>[\s-])\d(?P=sep) -> (?P=sep) recall group (?P<sep>[\s-])  
\b - only letters, nums and \_, but not +-@....  
([0]|[1-9]\d\*)(\.\d+)? vs ([0]|[1-9]\d\*\.?\d+)  
\w [a-zA-Z0-9\_] be careful for \_ !!!!!!  
(^|(?<=\s)) new line or space  
(^|\s) new line or space, but add the space to the result  
  
word = input()  
pattern = rf'\b{word}\b' # rf''

re.compile

email = input()

VALID\_DOMAINS = (".com", ".bg", ".net", ".org")  
regex\_domain = re.compile(r'\.[a-z]+')

if regex\_domain.findall(email)[-1] not in VALID\_DOMAINS:  
 print("Domain must be one of the following: .com, .bg, .org, .net")

word = input().casefold()  
pattern = rf'\b{word}\b' # -> how?  
# matches = re.findall(rf'(^|(?<=\s)){word}($|(?=\s))', text) # will not much HOW+?  
matches = re.findall(rf'\b{word}\b', text)  
print(len(matches))  
  
if there is more than 1 group, do not use re.findall(), but re.finditer or (?:...)  
(?:...) means do not create a group ID, but act as a group  
  
result = re.findall() # finds all, returns list  
result = re.search() # finds first, not iterable, returns match type or None  
re.match is anchored at the start  
re.fullmatch is anchored at the start and end of the pattern  
re.search is not anchored  
result = re.match() # finds first, if it's at the beginning only, but  
if re.search(pattern, names):  
 print("yes")  
else:  
 print("no")  
  
  
# pattern = r"\b(?P<Day>\d{2})([./-])(?P<Month>[A-Z][a-z][a-z])\2(?P<Year>\d{4})\b"  
pattern = r"\b(?P<Day>\d{2})(?P<sep>[\./-])(?P<Month>[A-Z][a-z][a-z])(?P=sep)(?P<Year>\d{4})\b"  
text1 = "13/Jul/1928, 10-Nov-1934, , 01/Jan-1951,f 25.Dec.1937 23/09/1973, 1/Feb/2016"  
dates = re.finditer(pattern, text1)  
# print(dates)  
for date in dates:  
 print(date)  
 num\_dict = date.groupdict() # Match into dict  
# print(f"Day: {num\_dict['Day']}, " # calling value of key=Day from num\_dict  
# f"Month: {num\_dict['Month']}, "  
# f"Year: {num\_dict['Year']}")  
# print(f"Day: {num[1]}, " # group(1) returns the group(1) Match  
# f"Month: {num[3]}, " # group(3) returns the group(3) Match  
# # f"Month: {num['Month']}" <=> f"Month: {num[3]}" -> both can be used  
# f"Year: {num['Year']}") # group(Year)(4) returns the group(Year)(4) Match  
# # f"Year: {num['Year']}" <=> f"Year: {num[4]}"  
# # -> both num['Year'] and num[4] can be used, because group4 is named Year  
 print(f"Day: {num\_dict['Day']}, Month: {num\_dict['Month']}, Year: {num\_dict['Year']}")  
 print(f"Day: {date['Day']}, Month: {date['Month']}, Year: {date['Year']}")  
 print(f"Day: {date[1]}, Month: {date.group(3)}, Year: {date[4]}")  
 # !!! use date.group(1) or date.group('Day'), but not date[1] or date['Day'],  
 # because it could NOT be available in next release!!!  
 print(date.group()) # group(0) returns the whole Match  
 print(date.group(1)) # group(1) returns Day  
 print(date.group('Month')) # group(2) returns 'Month'  
 print(date.groups()) # all groups as tuple ('13', '/', 'Jul', '1928')  
# dates1 = re.findall(pattern, text1)  
# print(dates1)  
# for date in dates1:  
# print(f"Day: {date[0]}, Month: {date[2]}, Year: {date[3]}")  
dates = re.match(pattern, text1) # MATCH IS NOT ITERABLE, searches at the BEGINNING ONLY  
print(dates) # match & search are same type, but the scope  
print(type(dates))  
print(dates.groupdict())  
dates = re.search(pattern, text1) # returns the same as match, BUT in ALL ROWS  
print(dates) # match & search are same type, but the scope  
print(type(dates))  
print(dates.groupdict())  
  
txt = "The rain in Spain"  
x = re.sub(r"\s", "9", txt, 2) # substitute(replace)  
print(x)  
  
txt = "The rain in Spain"  
x = re.split(r"\s", txt)  
print(x)  
  
text1 = input()  
text2 = input()  
text3 = input()  
pattern = r"\+359 2 \d{3} \d{4}\b|\+359-2-\d{3}-\d{4}\b"  
num1 = re.findall(pattern, text1) # more time  
num2 = re.findall(pattern, text2) # more time  
num3 = re.findall(pattern, text2) # more time  
regex\_pattern = re.compile(pattern)  
num11 = regex\_pattern.findall(text1) # faster  
num12 = regex\_pattern.findall(text2) # faster  
num13 = regex\_pattern.findall(text3) # faster  
  
print(\*res\_list, sep=', ')  
print(str\_res[:-2])

# Sets - кортежи(tuple) и множества(set)

**Unique unordered collection**

Sets can be used to perform mathematical set operations (union, intersection, symmetric difference, etc.)

usernames = set()

sorted(set(crafted))] => return list

[print(f"{toy}: {crafted.count(toy)}") for toy in sorted(set(crafted))]

1st\_Project\03\_Advanced\03\_Stacks\_Queues\_Tuples\_and\_Sets\_Exercise\Exercises\05\_santas\_present\_factory\_a.py

A set is a collection which is unordered and unindexed.

No repeated symbols.

Sets are written with braces curly brackets

text = "Hhello"   
set\_text = set(text)   
print(text) # Hhello  
print(set\_text) # {'H', 'o', 'e', 'l', 'h'}

a = set([1, 2, 3, 4])  
b = set([3, 4, 5, 6])  
print(a | b) # Union -> {1, 2, 3, 4, 5, 6}  
print(a & b) # Intersection -> {3, 4}  
print(a < b) # Subset -> False  
print(a > b) # Superset -> False  
print(a - b) # Difference -> {1, 2}  
print(a ^ b) # Symmetric Difference -> {1, 2, 5, 6}  
  
a.union(b) # Equivalent to a | b  
print(a.union(b)) # {1, 2, 3, 4, 5, 6}  
print(a) # {1, 2, 3, 4}  
a.intersection(b) # Equivalent to a & b  
a.issubset(b) # Equivalent to a <= b  
a.issuperset(b) # Equivalent to a >= b  
a.difference(b) # Equivalent to a - b  
a.symmetric\_difference(b) # Equivalent to a ^ b

a.update() updates the current set, by adding items from another set

def isdisjoint(self, \*args, \*\*kwargs):   
 *""" Return True if two sets have a null intersection. """*

The discard() method removes the specified item from the set. This method is different from the remove() method, because the remove() method will raise an error if the specified item does not exist, and the discard() method will not.

# Shortcuts

See PyCharm chapter

**Word:**

Ctrl + F6 – switch between open Word docs

Alt+ F7 - starts spell check in MS Word

# Slicing

[::] no beginning and end  
a = "2371"  
x = a[::-1] # 1732

x = a[:-1] # 1  
y = a[::-2] # 13  
z = a[::2] # 27

z = a[:2] # 23  
b = list(a) # ['2', '3', '7', '1']  
a = "0123456789"  
x1 = a[1::2] # 13579  
x2 = a[::2] # 02468  
x3 = a[::3] # 0369  
c = list(a)  
c.extend(b)  
d = a[1:7]  
a = [1, 2, 3, 4, 5, 6, 7]  
b = a[-5:-2] # new not referent  
b = a[-3:-6:-2] # [5, 3]  
b = a[:] # new not referent  
b = a[::] # new not referent  
txt = "Welcome To My World"  
x = txt[-5::] # World

x = txt[-5:] # World  
x = txt[14:] # World  
x = txt[slice(-5, len(txt), 1)] # World  
x = txt[slice(-5, 19, 1)] # World  
x = txt[-5::2] # Wrd  
x = txt[-5:2] # empty because 2 = -17  
x = txt[-17:9] # lcome T  
x = txt  
print(x)

# removing elements in the middle of the list

a\_nums = a\_nums[:left\_idx] + a\_nums[right\_idx + 1:]  
print(a\_nums)  
# =>  
for i in range(idx + value, idx - value - 1, -1):  
 b\_nums.pop(i)  
print(b\_nums)  
# =>  
del c\_nums[left\_idx:right\_idx + 1]  
print(c\_nums)

# Symbol names

= equal

{ open brace

( ) parenthesis

[ open bracket

% percent

? Question Mark

| pipe or bar

! "bang", "exclamation point"

@ "at", and rarely, "strudel"

# "crunch", "hash", "pound", and rarely, "octothorpe"

^ "circumflex", "hat", "chapeau"

& "ampersand", "and"

\* "splat", "star", "asterisk", "times" (as in multiplication)

\_ "underscore"

- "hyphen", "dash", "minus sign"

. "dot", "period"

, "comma"

: "colon"

; "semi-colon"

/ "slash"

\ "backslash"

~ "twiddle", also "squiggle", or more correctly, "tilde"

' "tick", "quote", "apostrophe"

" " double-quote"

` "backtick", "backquote"

< "less-than", "left angle bracket"

> "greater-than", "right angle bracket"

# Text

name = 'Test'  
print('name is: {}'.format(name)) # name is: Test  
print(f'name is: {name}') # name is: Test  
# print() is function  
# .format(name) is method

Python **integers** are **immutable**

Python **floats** are **immutable**

Python **strings** are **immutable**

This means that once a string is created,

it is **not** possible to **modify** it

name = 'George'  
name[0] = 'P' # Error не може да променим G

print(name) # George

name = 'Ime' # заделя друго място в паметта

различно от мястото за George  
print(name) # Ime  
name = 4 # заделя трето място в паметта  
print(name) # 4

string **interpolation** are string **literals**(буквален)

that allow **embedded**(вградени) expressions

name = 'Test New'  
print(name[:2]) # Te  
print(name[:3]) # Tes  
print(name[ ]) # Error   
print(name[3:]) # t New

print(name[2:6]) # st N

# creating new text with removed chars in the middle of the list

a\_nums = a\_nums[:left\_idx] + a\_nums[right\_idx + 1:]  
print(a\_nums)

# Time

time.sleep(2) #-> wait for 2 seconds (secs)

# region datetime timedelta, strptime, strftime

from datetime import datetime, timedelta

# input\_time = "8:00:00"  
input\_time = "2023:8:00:00:17" # Month is omitted   
current\_time = datetime.strptime(input\_time, "%Y:%H:%M:%S:%d")  
current\_time += timedelta(seconds=7)  
# class datetime.timedelta(days=0, seconds=0, microseconds=0, milliseconds=0, minutes=0, hours=0, weeks=0)  
print(current\_time.strftime("p[%H:%M:%S{q")) # p[08:00:07{q  
print(current\_time.strftime("%H:%M:%S-(%d/%Y)")) # 08:00:07-(17/2023) - Month is omitted

# endregion

# region Diff = End\_time - Start\_time

import time

start\_time = time.time()  
test\_list = [x for x in range(100000)]  
while test\_list:  
 test\_list.pop()  
diff = time.time() - start\_time  
print(diff)  
start\_time = time.time()  
test\_list = [x for x in range(100000)]  
while test\_list:  
 test\_list.pop(0)  
diff = time.time() - start\_time  
print(diff)

# endregion

# Tuples - кортежи(tuple) и множества(set)

t = (1, )   
t = (1, 2, 3)  
t = 1, 2, 3

nums = tuple(int(x) for x in input().split())

**two** available tuple methods

**count and index**

Tuples are **immutable objects,** but the **objects**, inside the tuples, **are mutable**

nums = [1, 2]  
my\_tuple = (nums, 7, 9) # tuple are immutable but variables are mutable  
print(my\_tuple) # ([1, 2], 7, 9)  
nums.append(3) # change NUMS in tuple!!! It will not work after redefining it in the next row  
nums = [1, 2, 29] # does not change NUMS in tuple!!! create new NUMS different from NUMS in tuple

print(my\_tuple) # ([1, 2, 3], 7, 9) -> variables inside the tuple are mutable  
my\_tuple[0][2] = 12 # if we want to access NUMS in tuple again  
my\_tuple[0].append(43) # if we want to access NUMS in tuple again  
print(my\_tuple) # ([1, 2, 12, 43], 7, 9) -> variables inside the tuple are mutable  
nums.append(23) # [1, 2, 29, 23]  
print(nums) # [1, 2, 29, 23]  
print(my\_tuple) # ([1, 2, 12, 43], 7, 9) -> variables inside the tuple are mutable

# ZZZ Other

## If… Else … replacement

even\_set.add(num) if num % 2 == 0 else odd\_set.add(num)

map\_function = {  
 1: lambda x: numbers.append(x[1]),  
 2: lambda x: numbers.pop() if numbers else None,  
 3: lambda x: print(max(numbers)) if numbers else None,  
 4: lambda x: print(min(numbers)) if numbers else None,  
} # There must be lambda x: on each Key: Value !!!  
for \_ in range(int(input())):  
 command = [int(x) for x in input().split()]  
 # map\_function[command[0]](command)  
 if map\_function.get(command[0]):  
 map\_function[command[0]](command)  
 else:  
 print("anything")  
 # try:  
 # map\_function[command[0]](command)  
 # except KeyError:  
 # print("anything")  
----------------------------------------------------------------------

from functools import reduce

map\_function = {  
 '+': lambda x: reduce(lambda a, b: a + b, x),  
 '-': lambda x: reduce(lambda a, b: a - b, x),  
 '/': lambda x: int(reduce(lambda a, b: a / b, x)),  
 # '/': lambda x: reduce(lambda a, b: a + b if a == 0 or b == 0 else a / b, x),  
 '\*': lambda x: reduce(lambda a, b: a \* b, x),  
}  
for el in data:  
 if el in map\_function:  
 res = map\_function[el](temp\_list)  
 else:  
 temp\_list.append(int(el))

---------------------------------------------------------------

map\_func = {  
 "Add First": lambda x: set1.update(x),  
 "Add Second": lambda x: set2.update(x),  
 "Remove First": lambda x: set1.difference\_update(x),  
 "Remove Second": lambda x: set2.difference\_update(x),  
 # "Check Subset": lambda x: print(set1.issubset(set2) or set2.issubset(set1))  
 **"Check Subset": lambda x: print("True") if set1.issubset(set2) or set2.issubset(set1) else print("False")**  
}  
for \_ in range(int(input())):  
 action1, action2, \*info = input().split()  
  
 map\_func[action1 + ' ' + action2](map(int, info))