# 1

def fact(x):

if x == 1 or x == 0:

return 1

else:

return fact(x - 1) \* x

# 2

def filter\_even(li):

return list(filter(lambda x: x % 2 == 0, li))

# 3

def square(li):

return list(map(lambda x: x 2, li))

# 4

def bin\_search(li, element):

l = 0

r = len(li) - 1

while r >= l:

mid = (r + l) // 2

if li[mid] > element:

r = mid - 1

elif li[mid] < element:

l = mid + 1

else:

return mid

return -1

# 5

def is\_palindrome(string):

string = ''.join(filter(lambda x: x.isalpha(), string))

start = 0

finish = len(string) - 1

while start != len(string) // 2:

if string[start].lower() != string[finish].lower():

return "NO"

start += 1

finish -= 1

return "YES"

# 6

OPS = {'+': lambda x, y: x + y, '-': lambda x, y: x - y,

'\*': lambda x, y: x \* y, '//': lambda x, y: x // y,

'%': lambda x, y: x % y, '': lambda x, y: x \*\* y}

def load\_file(filename):

file = open(filename, encoding='utf-8')

data = [line.strip().split(' ') for line in file.readlines()]

file.close()

return data

def evaluate(l\_number, r\_number, op):

if op in OPS:

return OPS[op](l\_number, r\_number)

else:

print('Неизвестная операция')

def calculate(path2file):

data = load\_file(path2file)

operation = 0

f\_operand = 1

s\_operand = 2

result = []

for exp in data:

result.append(str(evaluate(int(exp[f\_operand]), int(exp[s\_operand]), exp[operation])))

return ','.join(result)

# 7

def load\_file2(filename):

file = open(filename, encoding='utf-8')

data = [line for line in file.readlines()]

file.close()

return data

def substring\_slice(path2file\_1, path2file\_2):

data\_1 = load\_file2(path2file\_1)

data\_2 = load\_file2(path2file\_2)

result = []

for line\_1, line\_2 in zip(data\_1, data\_2):

start\_index, end\_index = map(int, line\_2.strip().split())

result.append(line\_1[start\_index:end\_index + 1])

return ' '.join(result)

# 8

import json

def decode\_ch(string\_of\_elements):

periodic\_table = json.load(open('periodic\_table.json', encoding='utf-8'))

search\_index = 1

result = ''

while string\_of\_elements:

last\_ch = string\_of\_elements[search\_index:search\_index + 1].isupper() \

if string\_of\_elements[search\_index:search\_index + 1] else True

if string\_of\_elements[:search\_index] in periodic\_table and last\_ch:

result += periodic\_table[string\_of\_elements[:search\_index]]

string\_of\_elements = string\_of\_elements[search\_index:]

search\_index = 1

else:

search\_index += 1

return result

# 9

class Student:

def *init*(self, name, surname, grades=None):

[self.name](https://self.name/) = name

self.surname = surname

self.fullname = name + ' ' + surname

if grades is None:

grades = [3, 4, 5]

self.grades = grades

def is\_otlichnik(self):

return 'Yes' if self.mean\_grade() >= 4.5 else 'NO'

def greeting(self):

return f'Hello, I am {self.fullname}'

def mean\_grade(self):

return sum(self.grades) / len(self.grades)

def *add*(self, other):

return f'{[self.name](https://self.name/)} is friends with {[other.name](https://other.name/)}'

def *str*(self):

return self.fullname

# 10

class MyError(Exception):

def *init*(self, msg):

self.msg = msg

super().*init*(msg)