1. problema1-KOTLIN

2. *###Ilizand genericile sa se implementeze in Kotlin o functie extensie pe un subarbore de tipuri de date (vezi ierarhia de tipuri de date cu limitare superioara din curs/lab) care sa verifice daca variabila contine un nr si apoi sa determine daca numarul este prim sau nu.###*

```
3.
4.  fun List<Number>.allPrimes():Boolean
5.  {
6.      return !this.any{it !is Int || !it.isPrime()}
7.  }
8.
9.  fun Int.isPrime():Boolean{
10.     if(this<0)
11.         return false
12.     if(this in 1..2)
13.         return true
14.     return !2.until(this/2+1).any{this%it==0}
15. }
16.
17. fun main()
18. {
19.     val lists=listOf(
20.         listOf(1,2,3,4,5),
21.         listOf(1.0,2.0,3.0),
22.         listOf(1,3,7,11))
23.     lists.forEachIndexed { index, list ->
24.         println("List $index: $list -> all are primes: ${list.allPrimes()}")
25.     }
26.
27. }
```

28. problema2-Python

29. *### Sa se implementeze in Python echivalentul unei grupari de fire(ThreadPool) utilizand modulul threading si care sa ne puna la dispozitie utilizand modelul comanda posibilitatea de a inspecta starea firului de executie, de a-l opri temporar sau definitiv sau de a-l porni.###*

```
30.
31. import threading
32. import time
33. from time import sleep
34. from copy import copy
35.
36. class MyThreadPool:
37.     def __init__(self,n:int):
38.         self.__threads=[threading.Thread() for _ in range(n)]
39.         self.__get_thread_state=lambda threads, i:threads[i].is_alive()
40.         self.__delay_thread=lambda threads, i, delay:None
41.         self.__start_thread=lambda threads, i:threads[i].start()
42.         self.__map=lambda func, *args:threading.Thread(target=func, args=args)
43.
44.     def __enter__(self):
45.         return self
46.
47.     def __exit__(self, exc_type, exc_value, tb):
48.         for thread in self.__threads:
49.             if thread.is_alive():
50.                 thread.join()
51.
52.     def start_thread(self,i):
53.         self.__start_thread(self.__threads,i)
54.
55.     def map(self, i, func, *args):
56.         if self.__threads[i].is_alive():
57.             self.__threads[i].join()
58.         self.__threads[i]=self.__map(func,args)
59.
60.     def delay_thread(self, i, delay):
61.         self.__delay_thread(self.__threads,i,delay)
62.
63.     def get_thread_state(self,i):
64.         return self.__get_thread_state(self.__threads,i)
65.
66. def print_lists(lists):
67.     for list in lists:
68.         for i in list:
69.             print(f"{i} ",end='')
70.             sleep(1)
71.
72.
73. def reverse_print_lists(lists):
74.     for list in lists:
75.         aux=copy(list)
76.         aux.reverse()
77.         for i in aux:
78.             print(f"{i} ",end='')
79.             sleep(1)
80.
81. if __name__ == '__main__':
82.     with MyThreadPool(3) as pool:
83.         list=[1,2,3,4,5]
```

```
        pool.map(0,print_lists,list)
        pool.start_thread(0)
        print(f"Thread 0 is alive: {pool.get_thread_state(0)}")
        pool.map(1,reverse_print_lists,list)
        pool.start_thread(1)
        print(f"Thread 1 is alive: {pool.get_thread_state(1)}")
        time.sleep(7)
        print(f"\nThread 0 is alive: {pool.get_thread_state(0)}")
        print(f"Thread 1 is alive: {pool.get_thread_state(1)}")
```

93.problema3-Python

94. *###Sa se scrie un program Py care va utiliza modelul mediator pentru a realiza o interactiune de tipul transmitere bidirectionala(fol cozi intre procese) intre studenti si profesori utilizand asistentul(un obiect probesor,unul asistent si mai multe obiecte student). Se deseneaza diagrame si se explica solid.###*

95.

```
from queue import Queue

class Student:
    def __init__(self, name, mediator):
        self.name = name
        self.mediator = mediator

    def send_message(self, message):
        self.mediator.send_message(self, message)

    def receive_message(self, sender, message):
        print(f"{self.name} a primit mesajul de la {sender}: {message}")

class Professor:
    def __init__(self, name, mediator):
        self.name = name
        self.mediator = mediator

    def send_message(self, message):
        self.mediator.send_message(self, message)

    def receive_message(self, sender, message):
        print(f"{self.name} a primit mesajul de la {sender}: {message}")

class Mediator:
    def __init__(self):
        self.queue = Queue()

    def send_message(self, sender, message):
        self.queue.put((sender, message))

    def start_processing(self):
        while not self.queue.empty():
            sender, message = self.queue.get()
            receiver = None

            # Determină destinatarul în funcție de tipul expeditorului
            if isinstance(sender, Student):
                receiver = professor
            elif isinstance(sender, Professor):
                receiver = student

            # Transmitere mesaj către destinatar
            if receiver:
                receiver.receive_message(sender.name, message)

# Exemplu de utilizare
mediator = Mediator()

student = Student("John", mediator)
professor = Professor("Profesorul", mediator)

mediator.student = student
mediator.professor = professor

student.send_message("Buna ziua!")
professor.send_message("Salutare!")

mediator.start_processing()

```

problema 4 - Python

```
###Intr-un fisier text cu minim 50 val (nr) se iau cate 10 val consecutive care vor fi procesate intr-un thread separat(din modulul Threading, Python) (deci se lanseaza in executie minim 5 thread-uri). Datele sunt depuse intr-un ADT, apoi se va efectua o procesare de tip lambda care va gasi minimul, maximul si media din secventa, apoi va extrage o submultime ce contine numai numerele care se afla in intervalul media+ sau -media patratica a secventei procesate.###

from math import sqrt
from functional import seq
from threading import Thread

min_fun = lambda list: seq(list).min()
max_fun = lambda list: seq(list).max()
mean_fun = lambda list: seq(list).sum() / seq(list).len()
square_mean_fun = lambda list: sqrt(seq(list).map(lambda it: it * it).sum() / seq(list).len())
subsequence_fun = lambda list: seq(list).filter(lambda it: -square_mean_fun(list) <= it <= square_mean_fun(list))
if __name__ == "__main__":
    lists = []
    numbers = seq(open("output.txt", "r").readline().split(' ')) \
        .map(lambda it: it.replace('\t', '')) \
        .map(lambda it: int(it))
    while numbers.len() > 0:
        if numbers.len() >= 10:
            lists.append(numbers.take(10).list())
            numbers = numbers.drop(10)
        else:
            lists.append(numbers.take(numbers.len()).list())
            numbers = numbers.drop(numbers.len())

    results = []  # min,max,mean,subsequence
    threads = []
    for list in lists:
        aux = Thread(target=lambda list, results: results.append(
            (min_fun(list), max_fun(list), mean_fun(list), subsequence_fun(list))), args=(list, results,))
        aux.start()
        threads.append(aux)
    for thread in threads:
        thread.join()

    i=0
    for i in range(len(lists)):
        print(f"Sequence:{lists[i]}, min:{results[i][0]}, max:{results[i][1]}, mean:{results[i][2]}, subsequence:{results[i][3]}")
    i = 0
```

problema 6-Kotlin

```
        Pornind de la exmplul din curs cu functori none si some, sa se creeze in Kptlin o transformare(functor) care se aplica asupra lui toInt.Astfel,daca este none se va inlocui cu -1, daca este some se genereaza o valoare aleatoare de zero sau 1).Apoi transformarea va fi aplicata cu map.###

sealed class TipSimplu<out T>{
    object None:TipSimplu<Nothing>(){
        override fun toString()="Cu None"
    }
    data class Some<out T>(val value:T):TipSimplu<T>()
    companion object
}


fun <T>TipSimplu<T>.toInt():Int=when(this){
    TipSimplu.None->-1
    is TipSimplu.Some->(0..1).random()
}

fun TipSimplu<Int>.map(transform:(TipSimplu<Int>)->Int):TipSimplu<Int>{
    return TipSimplu.Some<Int>(transform(this))
}

fun main()
{
    println(TipSimplu.None.map(TipSimplu<Int>::toInt))
    println(TipSimplu.Some(3).map(TipSimplu<Int>::toInt))
    println(TipSimplu.Some(5).map(TipSimplu<Int>::toInt))
}

---Problema 18
###Sa se creeze in Py un program care primeste un sir de cautare si mai multe nume de fisiere text(inclusiv sursa program).Aplicatia caua daca acest sir se gaseste, apoi afiseaza pentru fiecare fisier numerele liniilor unde se afla acesta.Programul primeste de la linia de comanda parametri de intrare, iar daca este nevoie, mai primeste si un sir de inlocuire a celui cautat, caz in care face search&replace.###

import re

def main():
```

```
    search_text=input("sirul care trebuie cautat=")
    number_of_files=int(input("numarul de fisiere="))
    fnames=[]
    for _ in range(number_of_files):
        file_name=input("nume fisier=")
        fnames.append(file_name)

    replace_flag=input("doresti sa inlocuiesti cu ceva textul(y/n)?\n").lower() == 'y'
    replace_text=""
    if replace_flag:
        replace_text=input("noul text=")

    for name in fnames:
        file_name_written=False
        replaced_lines=[]

        with open(name,"r") as file:
            lines=file.readlines()
            replaced_lines=lines.copy()

            for line in lines:
                if search_text in line:
                    if not file_name_written:
                        file_name_written=True
                        print(f"inside {name}")
                    index=lines.index(line)
                    print(f"at line {index}")
                    if replace_flag:
                        replaced_lines[index]=line.replace(search_text,replace_text)

        #this is bs
        if replace_flag:
            with open(name,"w") as file:
                file.writelines(replaced_lines)
main()
```

problema 7-Python

```
###Sa se implementeze in Py un automat cu 3 stari unde fiecare stare este gestionata de un proces in care sa se proceseze cu expresii lambda o lista de numere intregi trimisa ca argument in constructor astfel: in S0 se verifica daca mai sunt elemente in lista (daca da, se duce in starea S1 unde se identifica si se sterge primul element par gasit, apoi se trece in S2), in S2 se identifica si se sterge primul element impar gasit, apoi se trece in S0. La fiecare stergere se afiseaza elementul sters. In S0 daca nu mai exista elemente se anunta acest lucru, iar programul se termina.###
import time

from multiprocessing import Process, Queue
from functional import seq


def State0(q):
    l = q.get()
    if len(l) > 0:
        q.put(True)
        q.put(l)
    else:
        q.put(False)
    time.sleep(1)



def State1(q):
    l: list = q.get()

    if seq(l).filter(lambda it:it%2==0).len()>0:
        aux = seq(l).filter(lambda it: it % 2 == 0).first()
        l.remove(aux)
        print(f"State 1 a eliminat {aux}")
    q.put(True)
    q.put(l)
    # time.sleep(1)


def State2(q):
    l: list = q.get()
    if seq(l).filter(lambda it:it%2!=0).len()>0:
        aux = seq(l).filter(lambda it: it % 2 != 0).first()
        l.remove(aux)
        print(f"State 2 a eliminat {aux}")
    q.put(True)
    q.put(l)
    # time.sleep(1)


class FSM:
    def __init__(self, list):
        self.__queue = Queue()
        self.__queue.put(True)
        self.__queue.put(list)
        self.__states=[State0,State1,State2]
```

```
        self.__current_state = 0

    def start(self):
        while self.__queue.get():
            process = Process(target=self.__states[self.__current_state], args=(self.__queue,))
            process.start()
            self.__current_state = (self.__current_state + 1) % 3
            process.join()


if __name__ == "__main__":
    l = list(range(10))
    fsm = FSM(l)
    fsm.start()
```

problema 17 - Python - mamifere

###Utilizand modelul Pod sa se scrie in Py un prog care va porni de la un model mamifer si apoi va permite instantiere de obiecte de tip om,femeie,caine,pisica. Acestor obiecte li se vor adauga 3-4 functionalitati suplimentare la alegere utilizand decoratorii: fct care descrie interactiunea dintre om si femeie sau om si pisica etc. ###

```
from abc import ABCMeta, abstractmethod


class MamiferAbstraction(metaclass=ABCMeta):
    def __init__(self, implementation):
        self._implementation = implementation

    @abstractmethod
    def speak(self):
        pass

    @abstractmethod
    def preffered_food(self):
        pass


class Mamifer(MamiferAbstraction):
    def __init__(self, implementation):
        super().__init__(implementation)

    def speak(self):
        self._implementation.speak()

    def preffered_food(self):
        self._implementation.preffered_food()


class MamiferImplementation(metaclass=ABCMeta):
    @abstractmethod
    def speak(self):
        pass

    @abstractmethod
    def preffered_food(self):
        pass


class Femeie(MamiferImplementation):
    def speak(self):
        print("Vorbeste cu voce de femeie.")

    def preffered_food(self):
        print("Caviar")


# Am pus clasa barbat in loc de clasa om deoarece mi se pare cam ciudat sa ai ca si clase frati clasele Femeie si Om...
class Barbat(MamiferImplementation):
    def speak(self):
        print("Vorbeste cu voce de barbat.")

    def preffered_food(self):
        print("Mici")


class Caine(MamiferImplementation):
    def speak(self):
        print("Ham ham.")

    def preffered_food(self):
        print("Carne")


class Pisica(MamiferImplementation):
    def speak(self):
        print("Miau miau.")

    def preffered_food(self):
```

```
        print("Peste")

if __name__ == '__main__':
    caine=Mamifer(Caine())
    pisica=Mamifer(Pisica())
    femeie=Mamifer(Femeie())
    barbat=Mamifer(Barbat())
    mamifere=[caine,pisica,femeie,barbat]
    for mamifer in mamifere:
        mamifer.speak()
        mamifer.preffered_food()
```

problema lunga cu more_itertools, map_reduce, indexul invers pt un set de doc text

```
from more_itertools import flatten, map_reduce
from functional import seq


def read_words(files):
    words = []
    for file in files:
        aux = open(file).read().split(' ')
        aux = [(word, file) for word in aux]
        words.append(aux)
    words = list(flatten(words))
    return words


if __name__ == '__main__':
    files = ["file1.txt", "file2.txt"]
    words = read_words(files)
    key_func = lambda pair: pair[0]
    value_func = lambda pair: (pair[1], 1)
    reduce_func = lambda list_of_pairs: seq(list_of_pairs).reduce_by_key(lambda x, y: x + y)
    dict = dict(map_reduce(words, keyfunc=key_func, valuefunc=value_func, reducefunc=reduce_func))
    print('{')
    for pair in dict.items():
        print(f"\'{pair[0]}\' : {pair[1]}")
    print('}')
    i = 0
```

problema 25 - functie **lambda** asupra unei liste(colectii) si revenirea la o stare anterioara

###file1.csv

66,17,83,70,42,64,59,60,94,18,37,90,41,63,5,56,42,98,74,87,44,40,84,73,13,67,55,100,61,94,46,27,71,94,19,89,97,5,30,79,57,8,81,68,14,40,8,67,11,79,77,35,92,91,48,35,26,62,92,31,49,63,90,79,24,65,43,64,18,4,67,73,61,57,73,33,58,32,6,94,53,90,45,82,34,18,63,84,21,36,56,81,80,5,8,10,98,39,61,1

```
from functional import seq
class Originator:
    def __init__(self, l):
        self.__state = l

    def set_state(self, l):
        self.__state = l

    def get_state(self):
        return self.__state

    def save_state_to_memento(self):
        return Memento(self.__state)

    def restore_state_from_memento(self, memento):
        self.__state = memento.get_state()


class Memento:
    def __init__(self, state):
        self.__state = state

    def get_state(self):
        return self.__state


class Caretaker:
    def __init__(self):
        self.__states = []

    def add(self, state):
        self.__states.append(state)

    def get(self):
        rez = self.__states[-1]
        self.__states.remove(self.__states[-1])
```

```
        return rez


if __name__ == '__main__':
    numbers=open("file1.csv",'r').read().split(',')
    numbers=seq(numbers).map(lambda it:int(it)).list()

    f1=lambda x:(x%2==0 and x+1) or x
    f2=lambda x:3*x*x-2*x+1
    f3=lambda x:x+x+1
    functions=[f1,f2,f3]

    caretaker=Caretaker()
    originator=Originator(numbers)

    print(f"Lista originala: {numbers}")
    for i,f in enumerate(functions,1):
        caretaker.add(originator.save_state_to_memento())
        originator.set_state(seq(originator.get_state()).map(f).list())
        print(f"Lista dupa aplicarea functiei f{i}: {originator.get_state()}")
        restore=input("Doriti restaurarea starii anterioare a listei? 1-Da Altceva-Nu\nRaspunsul dumneavoastra: ")
        if restore=="Da":
            originator.restore_state_from_memento(caretaker.get())
    print(f"Lista finala: {originator.get_state()}")

```

problema 26-Py

###Secvente din Python(PyFunctional),corutinele(asyncio) si impartirea in mai multe bucati, sa se elimine dintr un fisier text rezultat din conversia unui epub spatiile multiple, salturile la linie noua.La sf va fi generat noul fisier.Se va folosi abordari de tipul impacheteaza-proceseaza-despacheteaza.###

```
from functional import seq
import asyncio
import re

coroutines_per_processing = 4


async def _elimin_spatii_multiple(partition: str):
    return re.sub("[ ]{2,}", ' ', partition)


async def elimin_spatii_multiple(text):
    # text = seq(open(file, 'r').readlines())
    partition_len = text.len() // coroutines_per_processing + 1
    partitions = []
    while text.len() > 0:
        partitions.append(text.take(partition_len).make_string(''))
        text = text.drop(partition_len)
    tasks = [asyncio.create_task(_elimin_spatii_multiple(partition)) for partition in partitions]
    new_text = ''.join([await task for task in tasks])
    return new_text


async def _elimin_salturi_linie_noua(partition: str):
    return partition.replace('\n', '')


async def elimin_salturi_linie_noua(text):
    partition_len = text.len() // coroutines_per_processing + 1
    partitions = []
    while text.len() > 0:
        partitions.append(text.take(partition_len).make_string(''))
        text = text.drop(partition_len)
    tasks = [asyncio.create_task(_elimin_salturi_linie_noua(partition)) for partition in partitions]
    new_text = ''.join([await task for task in tasks])
    return new_text


async def main():
    file = "economics_to_be_happier.txt"
    text = seq(open(file, 'r',encoding='utf-8').readlines())
    task = asyncio.create_task(elimin_spatii_multiple(text))
    new_text = await task
    print(f"Dupa eliminarea spatiilor multiple:\n{new_text}")
    task = asyncio.create_task(elimin_salturi_linie_noua(seq(new_text)))
    new_text = await task
    print(f"Dupa eliminarea salturilor la linie noua:\n{new_text}")
    with open("OUTPUT.txt", 'w') as output:
        output.write(new_text)


asyncio.run(main())
```

```
asyncio
------------------
import asyncio
#in loc de @asy.coroutine folosim async pt a declara o corutina
async def doSum(inRange):#folosim async pt a avea acces la functiile asyncio
print( "Incepem sa adunam {} numere".format(inRange) )
sum = 0
chart = 10000
for i in range(inRange):
sum = sum + i
# print( i);
if i > chart:
chart = chart + 100000
await asyncio.sleep(0)#asteptam (timp scris intre paranteze) unitati de timp cand i> chart
print( "Terminam de adunat {} numere in rezultatul {}".format(inRange, sum) )
return sum

async def main(queue):#tot o corutina
while len(queue) != 0:
if len(asyncio.all_tasks()) < 5:#un set de task-uri neterminate inca
a = loop.create_task( doSum(queue.pop()) )#creem un task nou axat pe functia doSum
await asyncio.sleep(0)
await asyncio.wait([a])#se ruleaza obiecte așteptate în iterabilul [a] simultan și se blocheaza până la condiția specificată de return_when.
# print(a.result())

if __name__ == '__main__':
queue = []
queue.append(20000000)
queue.append(100000)
queue.append(30044440)
queue.append(266660000)
queue.append(1066600)
queue.append(3333300)
queue.append(20888000)
queue.append(1444000)
queue.append(3004440)
queue.append(20888000)
queue.append(14666600)

loop = asyncio.get_event_loop()#returneaza loop-ul curent de lucru
loop.run_until_complete( main(queue) ) #loop-ul ruleaza pana cand se termina corutina main
# loop.close()
#"""Să se realizeze calculul simultan pentru patru valori diferite ale lui n luate dintr-o
#coadă de către patru corutine diferite (se va utiliza modulul asyncio)."""



---------------------------------------------------------------------------------------------------------------
```

automat_3stari_expresiilambda

```
---------------------------------------
from functional import seq

class StateZero:
def __init__(self,automat):
self.automat=automat
def operation(self):
if(seq(self.automat.lista).non_empty()):
print(self.automat.lista)
self.automat.current_state=self.automat.states[1]
else:
self.automat.keep_running=False
print("lista vida")

class StateOne:
def __init__(self,automat):
self.automat=automat
def operation(self):
if(seq(self.automat.lista).exists(lambda number:number%2==0)):
self.automat.lista.remove(seq(self.automat.lista).\
find(lambda number: number%2==0))
self.automat.current_state=self.automat.states[2]
class StateTwo:
def __init__(self,automat):
self.automat=automat
def operation(self):
if(seq(self.automat.lista).exists(lambda number: number%2==1)):
self.automat.lista.remove(seq(self.automat.lista).\
```

```
77.  find(lambda number: number%2==1))
78.  self.automat.current_state=self.automat.states[0]
79.
80.  class Automat:
81.  def __init__(self,lista_numere):
82.  self.lista=lista_numere
83.  self.states=[StateZero(self),StateOne(self),StateTwo(self)]
84.  self.current_state=self.states[0]
85.  self.keep_running=True
86.  def start(self):
87.  while self.keep_running:
88.  self.current_state.operation()
89.
90.  def main():
91.  automat=Automat([2,3,4,5,6,7,8])
92.  automat.start()
93.  main()
94.  ------------------------------------------------------------
```

automat_3stari_lambda_listanrintregi

```
95.  ------------------------------------------------------------

96.  from functional import seq
97.
98.  #asa arata functia de detectie a unui numar prim
99.  isPrime=lambda number: all(number%divider!=0 for divider in range(2,int(number/2)+1))
100. #detecteaza daca un numar nu este prim
101. isNotPrime=lambda number: any(number%divider==0 for divider in range(2,int(number/2)+1))
102.
103. class StateZero:
104. def __init__(self,automat):
105. self.automat=automat
106. def operation(self):
107. if(seq(self.automat.lista).non_empty()):
108. print(self.automat.lista)
109. self.automat.current_state=self.automat.states[1]
110. else:
111. self.automat.keep_running=False
112. print("lista vida")
113.
114. class StateOne:
115. def __init__(self,automat):
116. self.automat=automat
117. def operation(self):
118. if(seq(self.automat.lista).exists(isPrime)):
119. self.automat.lista.remove(seq(self.automat.lista).\
120. find(lambda number: all(number%divider!=0 for divider in range(2,int(number/2)+1))))
121. self.automat.current_state=self.automat.states[2]
122. class StateTwo:
123. def __init__(self,automat):
124. self.automat=automat
125. def operation(self):
126. if(seq(self.automat.lista).exists(lambda num: not isPrime(num))):
127. self.automat.lista.remove(seq(self.automat.lista).\
128. find(lambda number: any(number%divider==0 for divider in range(2,int(number/2)+1))))
129. self.automat.current_state=self.automat.states[0]
130.
131. class Automat:
132. def __init__(self,lista_numere):
133. self.lista=lista_numere
134. self.states=[StateZero(self),StateOne(self),StateTwo(self)]
135. self.current_state=self.states[0]
136. self.keep_running=True
137. def start(self):
138. while self.keep_running:
139. self.current_state.operation()
140.
141. def main():
142. automat=Automat([2,3,4,5,6,7,8])
143. automat.start()
144. main()
145. ------------------------------------------------------
146. camera_hostel_adt_decoratori
147. ------------------------------------------------------
148. class Camera():
149. def operation(self):
150. pass
151.
152. class ConcreteCamera(Camera):
153. def operation(self):
154. print('Camera nu dispune de bar')
155. # return "ConcreteCamera"
156.
157. class Decorator(Camera):
158. _camera: Camera = None
159. def __init__(self, camera: Camera, price):
160. self._camera = camera
161. self.alcohol_price = price
162. @property
```

```
def component(self):
return self._camera
def operation(self):
return self._camera.operation()

class ConcreteDecoratorA(Decorator):
def operation(self):
print(f'In bar au fost urmatoarele cheltuieli: {self.alcohol_price}')
# return "DecoratedCamera"

def generare_plata(component: Camera):
component.operation()

if __name__ == '__main__':
'''
Componenta
ComponentaConcreta == Camera
Decorator
DecoratorConcretA = Decorator specific, pot face A, B, C... dar sa aiba implementari diferite == Consum_Bar
'''

# camere concrete
camera1 = ConcreteCamera()
camera2 = ConcreteCamera()
camera3 = ConcreteCamera()

# inainte de a decora camerele cu bar
camere_hotel = [camera1, camera2, camera3]
for camera in camere_hotel:
generare_plata(camera)
print()

# decorez camera2 cu bar
camera_decorata1 = ConcreteDecoratorA(camera1, 24)
camera_decorata2 = ConcreteDecoratorA(camera2, 0)
camere_hotel[0] = camera_decorata1
camere_hotel[1] = camera_decorata2

for camera in camere_hotel:
generare_plata(camera)
-----------------------------------------------
comanda_student,colega
-----------------------------------------------
from enum import Enum

class Stari(Enum):
FERICIT = 1
DISPERAT = 2
MORT = 3
STIE_POPA_CA_UMBLI_DEZGROPAT = 4

class Command:
def execute(self, student):
pass

class Dispera(Command):
def execute(self, student):
student.stareSufleteasca(Stari.DISPERAT)

class Rupe(Command):
def execute(self, student):
student.stareSufleteasca(Stari.MORT)

class Maxima(Command):
def execute(self, student):
student.stareSufleteasca(Stari.STIE_POPA_CA_UMBLI_DEZGROPAT)

class Bucurie(Command):
def execute(self, student):
student.stareSufleteasca(Stari.FERICIT)

class Student:
def __init__(self):
self.stare = Stari.FERICIT
def stareSufleteasca(self, stare):
self.stare = stare
def printStare(self):
print("Saracul student se simte: " + self.stare.name)

class Colega:
def __init__(self, baiat):
self.student = baiat
self.comanda = Bucurie()
def setComanda(self,comanda):
self.comanda = comanda
def executa(self):
self.comanda.execute(self.student)

def main():
un_student = Student()
o_colega = Colega(un_student)
un_student.printStare()
```

```
o_colega.setComanda(Dispera())
o_colega.executa()
un_student.printStare()

o_colega.setComanda(Rupe())
o_colega.executa()
un_student.printStare()

o_colega.setComanda(Maxima())
o_colega.executa()
un_student.printStare()

if __name__ == '__main__':
main()

-----------------------------------------
```

comanda(model)_student,profesor

```
-----------------------------------------
from enum import Enum


class Stari(Enum):
FERICIT = 1
DISPERAT = 2
MORT = 3
STIE_POPA_CA_UMBLI_DEZGROPAT = 4


class Command:
def execute(self, student):
pass


class Dispera(Command):
def execute(self, student):
student.stareSufleteasca(Stari.DISPERAT)


class Rupe(Command):
def execute(self, student):
student.stareSufleteasca(Stari.MORT)


class Maxima(Command):
def execute(self, student):
student.stareSufleteasca(Stari.STIE_POPA_CA_UMBLI_DEZGROPAT)


class Bucurie(Command):
def execute(self, student):
student.stareSufleteasca(Stari.FERICIT)


class Student:
def __init__(self):
self.stare = Stari.FERICIT

def stareSufleteasca(self, stare):
self.stare = stare

def printStare(self):
print("Saracul student se simte: " + self.stare.name)


class Profesor:
def __init__(self, baiat):
self.student = baiat
self.comanda = Dispera()

def setComanda(self, comanda):
self.comanda = comanda

def executa(self):
self.comanda.execute(self.student)


if __name__ == '__main__':
un_student = Student()
prof = Profesor(un_student)
un_student.printStare()

prof.setComanda(Rupe())
prof.executa()
un_student.printStare()

prof.setComanda(Maxima())
```

```
prof.executa()
un_student.printStare()

prof.setComanda(Bucurie())
prof.executa()
un_student.printStare()
```

---

fabrica_de_fabrici_limba_buton_tkinter

```
import abc
from tkinter import *

class Displayer(Frame):
def __init__(self, root):

self.root = root
self.canvas = Canvas(root)

class Factory_Default(metaclass=abc.ABCMeta):
@abc.abstractmethod
def UIdisplay(self):
pass

class Factory1_English(Factory_Default):
def __init__(self, name):
self.name = name
def UIdisplay(self):
print(self.name)
self.root = Tk()
self.root.title("english factory")
self.disp = Displayer(self.root)
self.disp.button = Button(self.root, text="Button one", padx=20, pady=20).grid(row=0, column=1)
self.disp.button = Button(self.root, text="Button two", padx=20, pady=20).grid(row=0, column=2)
self.disp.button = Button(self.root, text="Button three", padx=20, pady=20).grid(row=0, column=3)
self.root.mainloop()

class Factory2_Romanian(Factory_Default):
def __init__(self, name):
self.name = name
def UIdisplay(self):
print(self.name)
self.root = Tk()
self.root.title("fabrica romaneasca")
self.disp = Displayer(self.root)
self.disp.button = Button(self.root, text="Buton unu", padx=20, pady=20).grid(row=0, column=1)
self.disp.button = Button(self.root, text="Buton doi", padx=20, pady=20).grid(row=0, column=2)
self.disp.button = Button(self.root, text="Buton trei", padx=20, pady=20).grid(row=0, column=3)
self.root.mainloop()

class LanguageFactory:
def createFactory(self, name):
if name == 'english':
return Factory1_English(name)
elif name == 'romanian':
return Factory2_Romanian(name)

if __name__=='__main__':
FabricaMare = LanguageFactory()
type = input("introduce ui language romanian/english")
FabricaMica = FabricaMare.createFactory(type)
FabricaMica.UIdisplay()
```

---

fatada_boombox_SOLID

```
class play:
'''Subsystem # 1'''
def play(self):
print("playing...")

class record:
'''Subsystem # 2'''
def record(self):
print("recording...")

class fastforward:
'''Subsystem # 3'''
def fastforward(self):
print("fast forwarding...")

class rewind:
'''Subsystem # 4'''
```

```
def rewind(self):
print("rewinding...")

class radio:
'''Subsystem # 5'''
def radio(self):
print("switching to radio...")

class volumeup:
'''Subsystem # 6'''
def volumeup(self):
print("volume level up...")

class volumedown:
'''Subsystem # 7'''
def volumedown(self):
print("volume level down...")

class batterystats:
'''Subsystem # 8'''
def batterystats(self):
print("100% of your life...")

class Boombox:
'''Facade'''
def __init__(self):
#initializare metode n shit
self.playing = play()
self.recording = record()
self.fastforwarding = fastforward()
self.rewind = rewind()
self.radio = radio()
self.volumeup = volumeup()
self.volumedown = volumedown()
self.batterystatus = batterystats()
def startplaying(self):
self.volumeup.volumeup()
self.playing.play()
def startrecording(self):
self.volumeup.volumeup()
self.recording.record()
self.playing.play()
def switchtoradio(self):
self.radio.radio()
self.playing.play()
def volumetoggle(self):
self.volumeup.volumeup()
self.volumedown.volumedown()
def batterystats(self):
self.batterystatus.batterystats()

if __name__ == "__main__":
music = Boombox()
music.startplaying()

music.startrecording()
music.batterystats()

music.volumetoggle()

music.switchtoradio()
```

---

flatten_inregistrare

```
inregistrare={'Nume':'Bula','Locatia':{'Oras':'Pocreaca','Tara':'ROM'},'hobi':['Manea','Bautura','Femei']}
def flatten_tabel(y):
iesire={}
def flatten(x,nume=''):
if type(x) is dict:
for a in x:
flatten(x[a],nume+a+'_')#recirsiv
elif type(x) is list:
i=0
for a in x:
flatten(a,nume+str(i)+'_')
i+=1
else:
iesire[nume[:-1]]=x#afisare totala la nume
flatten(y)
return iesire
listaCurata=flatten_tabel(inregistrare)
print(listaCurata)
-----------------------------------------------------------
grafic_calcul_tkinter
-----------------------------------------------------------
import tkinter as tk
```

```
if __name__=='__main__':
frame =tk.Tk()
frame.geometry("500x1500")
canvas=tk.Canvas(frame,width=1000,height=1000)
canvas.place(x=0,y=0)

n0=0
x=[]
y=[]
for i in range(21,50):
x.append(i)
s=n0
for j in range(0,i):
s=s+j
y.append(s)

print(x)
print(y)

T= tk.Text(frame, height=5,width=100)
T.insert(tk.END,"i=")
T.insert(tk.END, x)
T.insert(tk.END, "\nS=")
T.insert(tk.END, y)
T.pack()

for i in range(0,len(x)-2):
if(i%2):
canvas.create_line(x[i], y[i], x[i+1], y[i+1],fill='red',width=4)
else:
canvas.create_line(x[i], y[i], x[i + 1], y[i + 1],fill='blue',width=4)
frame.mainloop()
```

---

intertool_closure_fisiere_temporare

```
import itertools as it
#closure ca un contor
def fabricaDeContoare(prefix):
iterator=it.count(start=0,step=1)
contorvar=next(iterator) #0
def contor():
nonlocal iterator,contorvar
contorvar=next(iterator)
nume1=prefix+str(contorvar)
nume2 =prefix + str(contorvar+1)
creeazaFisier(nume1,nume2)
return contorvar
def valoareCurentaContor():
nonlocal iterator,contorvar
return contorvar
def creeazaFisier(nume1,nume2):
numecomplet=nume1+"-index-"+nume2+".tmp"
f=open(numecomplet,"w+")
f.write("%d" %(valoareCurentaContor()))
f.close()
def numeUrmator():
nume1 = prefix + str(contorvar+1)
nume2 = prefix + str(contorvar + 2)
return nume1 + "-index-" + nume2 + ".tmp"
return contor,valoareCurentaContor,numeUrmator
contor,valoareContor,nextName=fabricaDeContoare("s")
print(contor())
print(contor())
print(contor())
print(valoareContor())
print(nextName())
print(contor())
#SV32Pyt356
```

---

lant_responsabilitati_nivelalerta_politie_nato

```
import abc
class Handler(metaclass=abc.ABCMeta):
def __init__(self,successor=None):
self.succesor=successor
def handle(self,request):
res=self.check_alertlevel(request)
if not res and self.succesor:
```

```
570.  self.succesor.handle(request)
571.  @abc.abstractmethod
572.  def check_alertlevel(self,request):
573.  """verificam nivelul de alerta"""
574.  class NATOHandler(Handler):
575.  @staticmethod
576.  def check_alertlevel(request):
577.  if request.find('0')!=-1:
578.  print("cererea {} tratata la NATO".format(request))
579.  return True
580.  else:
581.  return False
582.  class CSATHandler(Handler):
583.  def check_alertlevel(self,request):
584.  if request.find('1')!=-1:
585.  print("cererea {} tratata la CSAT".format(request))
586.  return True
587.  else:
588.  return False
589.  class SIEHandler(Handler):
590.  def check_alertlevel(self,request):
591.  if request.find('2')!=-1:
592.  print("cererea {} tratata la SIE".format(request))
593.  return True
594.  else:
595.  return False
596.  class SRIHandler(Handler):
597.  def check_alertlevel(self,request):
598.  if request.find('3')!=-1:
599.  print("cererea {} tratata la SRI".format(request))
600.  return True
601.  else:
602.  return False
603.  class PoliceHandler(Handler):
604.  def check_alertlevel(self,request):
605.  if request.find('4')!=-1:
606.  print("cererea {} tratata la Politie".format(request))
607.  return True
608.  else:
609.  return False
610.  class GuardHandler(Handler):
611.  def check_alertlevel(self,request):
612.  if request.find('5')!=-1:
613.  print("cererea {} tratata la paznicii muzeului".format(request))
614.  return True
615.  else:
616.  return False
617.  class FallbackHandler(Handler):
618.  @staticmethod
619.  def check_alertlevel(request):
620.  print("Am terminat de parcurs lantul - nu exista tratare pt cazul {}".format(request))
621.  return False
622.  if __name__ == '__main__':
623.  hg=GuardHandler()#creez gestionari
624.  hp=PoliceHandler()
625.  hsri=SRIHandler()
626.  hsie=SIEHandler()
627.  hcsat=CSATHandler()
628.  hnato=NATOHandler(FallbackHandler())
629.  hg.succesor=hp#creez lantul
630.  hp.succesor=hsri
631.  hsri.succesor = hsie
632.  hsie.succesor = hcsat
633.  hcsat.succesor = hnato
634.  requests=[
635.  "Sun la 4 ca mi s-a furat masina!",
636.  "Tre sa zic la 0 ca Rusia ne ataca",
637.  "Esti 3-ist frate!",
638.  "Suna la 5 ca s-a spart vasul din vitrina!",
639.  "1 nu faci nimic?",
640.  "Ca si sri tu esti 2 nu?",
641.  ]
642.  for request in requests:
643.  hg.handle(request)
644.  #sv108py334
```

---

mamifer_decorator

```
645.  from enum import Enum
646.  class Mamifer:#Component
647.  def __init__(self):
648.  print("Am creat un manifer")
649.
650.  class Femeie(Mamifer):#ConcreteComponent
```

```
def __init__(self):
print("Am creat o femeie.")
def ceva(self):
print("Femeia a facut ceva.")

class Om(Mamifer):#ConcreteComponent
def __init__(self):
print("Am creat un om.")
#Decorator
class Locare(Mamifer):
def __init__(self,mamifer):
self.mamifer = mamifer
def ceva(self):
self.mamifer.ceva()
print("Am pus mamiferul la locul lui.")

class MamiferHolder:
def __init__(self,mamifer):
self.local_mamifer = mamifer
self.local_mamifer.ceva()
self.decorated = Locare( self.local_mamifer)
self.decorated.ceva()

if __name__ == '__main__':
a = MamiferHolder(Femeie())
```

---

mamifer2_decorator

```
from enum import Enum
class Decorator:
def addDecorabil(self, mamifer):
self.decorat = mamifer
def interactiune(self):
pass

class Mamifer:
pass

class AnimalDeCompanie(Mamifer):
pass

class Caine(AnimalDeCompanie):
def __init__(self):
print("Ham!")

class Pisica(AnimalDeCompanie):
def __init__(self):
print("Meow!")

class Femeie(Mamifer, Decorator):#concreteDecorator
def __init__(self):
self.animal = AnimalDeCompanie()
print("Am creat o femeie.")

def amAnimal(self, animal):
self.animal = animal

def interactiune(self):
if isinstance(self.decorat, Om):#daca decorat este instanta a clasei Om
print("Femeie interactiune cu Om")

class Om(Mamifer):
def __init__(self):
self.animal = AnimalDeCompanie()
print("Am creat un om.")
def amAnimal(self, animal):
self.animal = animal

if __name__ == '__main__':
femeie = Femeie()
femeie.amAnimal(Caine())
femeie.addDecorabil(Om())
femeie.interactiune()
```

---

mediator_furnica_SOLID

```
import java.io.File
val procesare={ str:String-> str.substring(str.length/2-1,(str.length/2)+1)}
val stergere={str:String-> str.substring(2)}
fun main()
{
val filename="Text.txt"
```

```
val file = File(filename)
var exists=file.exists()
if(!exists)
{
println("Fisierul nu exista.Se va crea unul nou")
file.createNewFile()
file.writeText("Salut Lume!\nPlec din aceasta lume.")
}
var cuvinte:MutableList<String> = file.readText().split(" ",".","!","\n",",").toList() as MutableList<String>
cuvinte.removeIf {
it=="" //eliminam spatii libere
}
println(cuvinte)

var final=cuvinte.filter { it.length>3 }.map{procesare(it)}//pt kot39
println(final)
var final2= cuvinte.filter { it.length>3 }.map{stergere(it)}//pt Kot 40
println(final2)
}
#sv33py341
```

---

meidator_student_profesor

```
from abc import ABC
import multiprocessing as mp

message_queue = mp.Queue()

class Mediator(ABC):
def notify(self, sender, event):
pass

class ConcreteMediator(Mediator):
def __init__(self, student, profesor):
self.student = student
self.student.mediator = self#se apeleaza mediator.setter
self.profesor = profesor
self.profesor.mediator = self
def notify(self, sender, event):
if event == "Give messages to students":
if message_queue.empty() is True:
print("Studentii nu au primit mesaje.\n")
else:
print("Mesajele primite de la profesor sunt:")
while message_queue.empty() is False:
print(message_queue.get())
print()
elif event == "Give messages to teacher":
if message_queue.empty() is True:
print("Profesorul nu a primit mesaje.\n")
else:
print("Mesajele primite de la studenti sunt:")
while message_queue.empty() is False:
print(message_queue.get())
print()

class BaseComponent:
def __init__(self, mediator: Mediator = None):
self._mediator = mediator
@property
def mediator(self):
return self._mediator
@mediator.setter
def mediator(self, mediator):
self._mediator = mediator

class Student(BaseComponent):
def get_message(self):
self.mediator.notify(self, "Give messages to students")
def say_message(self, message):
message_queue.put(message)

class Profesor(BaseComponent):
def get_message(self):
self.mediator.notify(self, "Give messages to teacher") #se apeleaza @property
def say_message(self, message):
message_queue.put(message)

if __name__ == "__main__":
student1 = Student()
student2 = Student()
student3 = Student()
profesor = Profesor()
mediator = ConcreteMediator(student1, profesor)
mediator = ConcreteMediator(student2, profesor)
```

```
mediator = ConcreteMediator(student3, profesor)

profesor.get_message()
student2.get_message()

student1.say_message("Eu sunt Feri")
student1.say_message("Glumeam. Io-te feedbeck")
student2.say_message("Why are you running")
student3.say_message("Imi place pisicile")

profesor.get_message()
profesor.say_message("Multumesc pentru feedbeck")
profesor.say_message("Da")

student1.get_message()
```

---

multiprocessing_ciclica_cuvant_fisier_cozi

```
import multiprocessing

def worker(q):
print("Sunt procesul %s si am primit %s " %(multiprocessing.current_process().name,q.get()))

if __name__=='__main__':
f=open("in.txt","r")
a=f.read().split()

queue = multiprocessing.Queue()

p1 = multiprocessing.Process(target=worker, args=(queue,))
p1.start()

p2 = multiprocessing.Process(target=worker, args=(queue,))
p2.start()

p3 = multiprocessing.Process(target=worker, args=(queue,))
p3.start()

for i in a:
queue.put(i)
queue.close()
queue.join_thread()
p1.join()
p2.join()
p3.join()
#sv65 py378
```

---

multiprocessing_starmap_iterator

```
import multiprocessing


def count(iterator, k=1):
def f(x, k):
return x + k

contor = 0
for _ in iterator:
contor = f(contor, k)
return contor


if __name__ == '__main__':
nume_fisier = 'test.txt'
fisier_citit = []
with open(nume_fisier) as f:
linie = f.readline()
while linie:
fisier_citit.append(linie)
linie = f.readline()

k = 5 # numar procese
numar_linii_per_bucata = len(fisier_citit) // k + 1 # fiecare proces primeste (nr linii / numar procese) linii
numar_linii_pe_ultima_bucata = len(fisier_citit) - (k - 1) * numar_linii_per_bucata # ultimul proces primeste acelasi numar de linii
ca celelalte + orice a ramas
bucati = [[] for _ in range(k)]
i_bucata = -1 # la linia 30/31 valoarea se va schimba la 0, fiind prima linie executata din for
for i in range(len(fisier_citit)):
if i < numar_linii_per_bucata * (k-1):
```

```
if i % numar_linii_per_bucata == 0:
i_bucata += 1
bucati[i_bucata].extend(fisier_citit[i].split()) # fiecare bucata reprezinta o lista de cuvinte
else:
bucati[k - 1].extend(fisier_citit[i].split()) # ultima bucata

bucati_starmap = list(map(lambda bucata: (bucata,), bucati)) # iteram prin fiecare bucata din bucati si o inlocuim cu tupla formata doar din acea bucata
# e necesar pentru starmap

with multiprocessing.Pool(processes=k) as pool:
results = pool.starmap(count, bucati_starmap)
with open('result.txt', 'w') as f:
f.write(str(sum(results)))
#examenAdrianCristea
```

---

pod_bridge_desena_triunghi_cerc

```
import tkinter as tk

class DrawApi:#Implementor
def desenare(self,canvas):
pass
class Dreptunghi(DrawApi):#ConcreteImplementor
def desenare(self,canvas):
canvas.create_rectangle(400,200,100,100,fill="blue")

class Cerc(DrawApi):#ConcreteImplementor
def desenare(self,canvas):
canvas.create_oval(90,90,300,300,fill="yellow")

class Triunghi(DrawApi):#ConcreteImplementor
def desenare(self,canvas):
canvas.create_polygon((0, 100, 50, 0, 100, 100), fill="red")

class Shape():#Abstraction
def __init__(self,drawApi):
self.drawApi=drawApi
def Draw(self,canvas):
self.drawApi.desenare(canvas)

if __name__=='__main__':
frame = tk.Tk()
frame.geometry("600x600")
canvas = tk.Canvas(frame, width=600, height=600)
canvas.place(x=0, y=0)

dreptunghi=Shape(Dreptunghi())
cerc=Shape(Cerc())
triunghi=Shape(Triunghi())

dreptunghi.Draw(canvas)
cerc.Draw(canvas)
triunghi.Draw(canvas)

frame.mainloop()
#sv76 py314
```

---

coada_circulara_py_student,colega_kot

```
Subiect 1 - Python:
Să se implementeze în Python o coadă circulară de procese care scriu
într-un fișier. Procesele pot fi puse în așteptare sau pornite (se vor
utiliza decoratori). Pentru scrierea în fișier se va asigura coerența
utilizând semafoare (se poate crea tot un decorator sau nu).


*lipseste partea de wait, al doilea decorator
###################################
import random
import multiprocessing as mp
from time import sleep

semafor = mp.Semaphore(1)


```

```
def scrie():
with open("test.txt", "a") as file:
x: str = "Scriu un numar random: "
x += str(random.randint(1, 100))
x += '\n'
with semafor:
print("Am scris")
file.write(x)
sleep(1)
print("Am dormit")
print(x)


class CircularQueue(mp.Process):

def __init__(self, size):
self.size = size
self.queue = [None for i in range(size)]
self.front = self.rear = -1
mp.Process.__init__(self)

def enqueue(self, data):

if ((self.rear + 1) % self.size == self.front):
print(" Coada plina\n")
elif (self.front == -1):
self.front = 0
self.rear = 0
self.queue[self.rear] = data
else:
self.rear = (self.rear + 1) % self.size
self.queue[self.rear] = data

def dequeue(self):
if (self.front == -1):
print("Coada goala\n")
elif (self.front == self.rear):
temp = self.queue[self.front]
self.front = -1
self.rear = -1
return temp
else:
temp = self.queue[self.front]
self.front = (self.front + 1) % self.size
return temp

def display2(self):
for x in self.queue:
if x is not None:
pornire(x)
print()


def porneste(func):
def wrapper(*args, **kwargs):
output = func(*args, **kwargs)
output.start()
return output

return wrapper


@porneste
def pornire(p: mp.Process):
print("Am pornit ", p.__str__(), '\n')
return p


if __name__ == '__main__':
print("hello")
ob = CircularQueue(5)
p1 = mp.Process(target=scrie)
p2 = mp.Process(target=scrie)
ob.enqueue(p1)
ob.enqueue(p2)
ob.display2()

########################################


```

Subiect 2 - Kotlin:

Utilizând modelul comandă să se scrie în Kotlin un program care pornind de la o clasă student va asigura posibilitatea unor comenzi specifice unui obiect colegă care să conducă la schimbarea stării interne a unui obiect student (de exemplu din fericit în disperat). Apoi acestea vor fi utilizate într-un automat static (cu două stări fericit-nefericit) implementat utilizând modelul stare. Se vor desena diagrama de clase și de obiecte. Se va explica maniera de aplicare a principiilor SOLID.

```
##########################################
/*
* Respectarea principiilor SOLID
*
* Responsabilitate unica -> Codul poate fi organizat pe bucati mai mici, si anume stari, fiecare
* clasa comanda face doar un singur lucru
*
* Inchis/Deschis -> Codul poate fi modificat pentru extensii, adaugandu-se noi stari sau noi
comenzi
*
*/
```

```
1030. class Student{
1031. var state: StareStudent
1032. val fericit = StareStudentFericit(this)
1033. val disperat = StareStudentDisperat(this)
1034.
1035. init {
1036. state = fericit
1037. }
1038.
1039. override fun toString(): String {
1040. return "Student $state \n"
1041. }
1042.
1043. }
1044.
1045. abstract class StareStudent(var student: Student){
1046. open fun schimba() : Unit {}
1047. }
1048.
1049. class StareStudentFericit(student: Student) : StareStudent(student){
1050. override fun schimba() {
1051. student.state = student.disperat
1052. }
1053.
1054. override fun toString(): String {
1055. return "Fericit"
1056. }
1057. }
1058.
1059. class StareStudentDisperat(student: Student) : StareStudent(student){
1060. override fun schimba() {
1061. student.state = student.fericit
1062. }
1063.
1064. override fun toString(): String {
1065. return "Disperat"
1066. }
1067. }
1068.
1069. interface Colega{
1070. fun schimba(student: Student)
1071. }
1072.
1073. class SchimbaStare: Colega{
1074. override fun schimba(student: Student) {
1075. student.state.schimba()
1076. }
1077.
1078. }
1079.
1080. fun main(args: Array<String>) {
1081. val student1 : Student = Student()
1082. val colega : SchimbaStare = SchimbaStare()
1083. print(student1)
1084. colega.schimba(student1)
```

```
print(student1)
colega.schimba(student1)
print(student1)
}


##############################################3
```

---

proxy_pret_login

```
import datetime
class Hacker(Exception):
pass

class Observator:#Observer
def primesteRata(self, rata):
raise NotImplemented

class Pret(Observator):#ConcreteObserver
def __init__(self, initial):
self.initial = initial
def primesteRata(self, rata):
self.initial = self.initial - ((rata * self.initial) / 100)
return self.initial

class Dispatcher:
def __init__(self):
self.observatori = []
self.logger = Logger("Log.txt")
def addObs(self, obs):
self.observatori.append(obs)
def removeObs(self, obs):
self.observatori.remove(obs)
def trimiteNouaRata(self, rata, user):
for obs in self.observatori:
pretNou = obs.primesteRata(rata)
self.logger.log(user.user, rata, pretNou)

class Cont:
def __init__(self, user, parola):
self.user = user
self.parola = parola

class Service:#clasa Subject
def operatie(self):
raise NotImplemented

class IntroducereRata(Service):
def operatie(self):
return int(input('Rata noua: '))

class Proxy(Service):
def __init__(self):
self.service = None
self.cont = None
self.dispatcher = Dispatcher()
def operatie(self):
return self.service.operatie()
def checkAcces(self):
self.service = Logare()
if self.cont is None:
self.cont = self.operatie()
self.dispatcher.addObs(Pret(int(input('Pret initial: '))))
else:
nouBaiat: Cont = self.operatie()
if not (nouBaiat.user == self.cont.user and nouBaiat.parola == self.cont.parola):
print("Hacker-ule...iesi afara!")
raise Hacker
else:
self.service = IntroducereRata()
rata = self.operatie()
self.dispatcher.trimiteNouaRata(rata, self.cont)

class Logare(Service):#realSubject
def operatie(self):
return Cont(str(input('User: ')), str(input('Parola: ')))

class Logger:
def __init__(self, fila):
self.fisier = open(fila, 'a')
def close(self):
self.fisier.close()
def log(self, user, rata, pret):
```

```
self.fisier.write(
'User-ul ' + user + ' la ' + str(
datetime.datetime.now()) + ' a modificat pretul cu rata de ' + str(rata) + '%, noul pret devenind: ' + str(pret) + '\n')

if __name__ == '__main__':
proxy = Proxy()
while True:
proxy.checkAcces()
#sv50py335
```

---

search_sir_replace

```
import re

def main():
search_text=input("sirul care trebuie cautat=")
number_of_files=int(input("numarul de fisiere="))
fnames=[]
for _ in range(number_of_files):
file_name=input("nume fisier=")
fnames.append(file_name)

replace_flag=input("doresti sa inlocuiesti cu ceva textul(y/n)?\n").lower() == 'y'
replace_text=""
if replace_flag:
replace_text=input("noul text=")

for name in fnames:
file_name_written=False
replaced_lines=[]

with open(name,"r") as file:
lines=file.readlines()
replaced_lines=lines.copy()

for line in lines:
if search_text in line:
if not file_name_written:
file_name_written=True
print(f"inside {name}")
index=lines.index(line)
print(f"at line {index}")
if replace_flag:
replaced_lines[index]=line.replace(search_text,replace_text)

#this is bs
if replace_flag:
with open(name,"w") as file:
file.writelines(replaced_lines)
main()
```

---

strategy_erori_solid

```
import sys

class Warning(Exception):
pass

class Error(Exception):
pass

class GraveError(Exception):
pass


class Strategie:
file = None
fileGrav = None
fileWarning = None
@classmethod
def fisiereDeschidere(cls):
fisier_de_log = "Erori.txt"
cls.file = open(fisier_de_log, "w")
fisier_de_log_rau = "EroriGrave.txt"
cls.fileGrav = open(fisier_de_log_rau, "w")
fisier_de_log_mai_bun = "Warning.txt"
cls.fileWarning = open(fisier_de_log_mai_bun, "w")
```

```
def handleError(self):
pass
@classmethod
def inchidereFisierCaEAcademic(cls):
cls.file.close()
cls.fileGrav.close()
cls.fileWarning.close()

class StrategieEroare(Strategie):
def handleError(self):
print("A aparut o eroare!")
self.file.write("Chiar a aparut o eroare!\n")

class StrategieEroareGrava(Strategie):
def handleError(self):
self.fileGrav.write("Chiar a aparut o eroare grava!\n")
Strategie.inchidereFisierCaEAcademic()
sys.exit()

class StrategieWarning(Strategie):
def handleError(self):
self.fileWarning.write("Ai picat la PC ca ai warning!\n")


class Handler:
def __init__(self):
self.strategie = Strategie()
Strategie.fisiereDeschidere()
def setareStrategie(self, strategie):
self.strategie = strategie
def gestionam(self):
self.strategie.handleError()

def main():
handler = Handler()
while True:
print("Ce eroare vrei?")
print(" 0 - Grava")
print(" 1 - Normala")
print(" 2 - Doar warning")
opt = int(input(""))
try:
if opt == 0:
raise GraveError
elif opt == 1:
raise Error
else:
raise Warning
except GraveError:
handler.setareStrategie(StrategieEroareGrava())
handler.gestionam()
except Error:
handler.setareStrategie(StrategieEroare())
handler.gestionam()
except Warning:
handler.setareStrategie(StrategieWarning())
handler.gestionam()


if __name__ == '__main__':
main()
#sv46 py339
```

---

strategie_bancuri

Varianta 1

```
from random import choice


class Dictionary:
def __init__(self, file_name):
self.joke_book = {
"*": [],
"**": [],
"***": []
}
with open(file_name, "r") as file:
for line in file.readlines():
self.joke_book[line.split()[0]].append(line.strip())

def get_joke(self, joke_rank):
return choice(self.joke_book[joke_rank])


```

```
class GoodJokeStrategy:
def __init__(self, joke_collection):
self.joke_collection = joke_collection

def get_joke(self):
return self.joke_collection.get_joke("*")


class VeryGoodJokeStrategy:
def __init__(self, joke_collection):
self.joke_collection = joke_collection

def get_joke(self):
return self.joke_collection.get_joke("**")


class TheBestJokeStrategy:
def __init__(self, joke_collection):
self.joke_collection = joke_collection

def get_joke(self):
return self.joke_collection.get_joke("***")


class Student:
def __init__(self, joke_book):
self.joke_book = joke_book
self.strategy = GoodJokeStrategy(self.joke_book)

def get_feedback(self, feedback):
if feedback == "sad":
self.strategy = TheBestJokeStrategy(self.joke_book)
if feedback == "serious":
self.strategy = VeryGoodJokeStrategy(self.joke_book)
if feedback == "happy":
self.strategy = GoodJokeStrategy(self.joke_book)

def tell_a_joke(self):
return self.strategy.get_joke()

def listen(self, joke):
print(f"i listen to {joke}")
joke_rank = joke.split()[0]

if joke_rank == "*":
return "sad"
if joke_rank == "**":
return "serious"
if joke_rank == "***":
return "happy"


def main():
joke_book = Dictionary("jokes.txt")
student1 = Student(joke_book)
student2 = Student(joke_book)

feedback = student2.listen(student1.tell_a_joke())
print(feedback)
student1.get_feedback(feedback)
feedback = student2.listen(student1.tell_a_joke())
student1.get_feedback(feedback)
print(feedback)


main()


```

Varianta2

```
import abc
import random
#Se scrie un program Python care utilizand modelul strategie va alege in fuectie de reactia
#unui coleg un banc dintr-un dictionar care are seturi de vancuri bune, foarte bune si cele mai
#bune ( un fisier text care are un banc pe paragraf si la inceput are una/doua/trei stelute
#(codificarea calitatii vancului). Deci este nevoie de o clasa student, o metoda de a spune
#bancuri si una de a asculta si furniza o reactie.

def citimGlume(text):#metoda de as spune bancuri
bancuriBune = []
bancuriFoarteBune = []
bancuriCeleMaiBune = []

default_data = { 'ok': bancuriBune,
'bune': bancuriFoarteBune,
'foartebune':bancuriCeleMaiBune
```

```
}

glume = text.split("\t")#impartim prin TAB-uri
for element in glume:
if "***" in element:
bancuriCeleMaiBune.append( element)#adaugam in vectorul din dictionar
elif "**" in element:
bancuriFoarteBune.append( element)
elif "*" in element:
bancuriBune.append(element)


#print(default_data['ok'])
#print(default_data['bune'])
#print(default_data['foartebune'])
return default_data

class StrategieGlume( metaclass= abc.ABCMeta):#Strategy
@abc.abstractmethod
def spuneGluma(self):
pass

class StrategieGlumeOk(StrategieGlume):#ConcreteStrategy
def __init__(self, _dictionar):
self.dictionar = _dictionar

def spuneGluma(self):
listos = self.dictionar['ok']
gluma = listos[random.randint(0, len(listos)-1)]
print(gluma)
return gluma

class StrategieGlumeBune(StrategieGlume):#ConcreteStrategy
def __init__(self, _dictionar):
self.dictionar = _dictionar

def spuneGluma(self):
listos = self.dictionar['bune']
gluma = listos[random.randint(0, len(listos) - 1)]#alegem random o gluma
print(gluma)#si o listam
return gluma

class StrategieGlumeFoarteBune(StrategieGlume):#ConcreteStrategy
def __init__(self, _dictionar):
self.dictionar = _dictionar

def spuneGluma(self):
listos = self.dictionar['foartebune']
gluma = listos[random.randint(0,len(listos) -1) ]
print( gluma)
return gluma

class Student():#O clasa context
def __init__(self, _strategie):
self.strategie = _strategie

def spuneGluma(self):
return self.strategie.spuneGluma()#apeleaza metoda din Clasele ce extind StrategieGlume

def ascultaGluma(self,gluma):
if "***" in gluma:
print("hahaHAHAHAHAHAHA")#dupa ce se spune gluma,se asculta
elif "**" in gluma:
print("hahaHAHA")
elif "*" in gluma:
print("ha")

if __name__ == '__main__':
f = open("bancuri.txt", "r")
text = f.read()

dictionarGlume = citimGlume(text)
#print(glume)
strategie1 = StrategieGlumeOk(dictionarGlume);
strategie2 = StrategieGlumeFoarteBune(dictionarGlume);

student1 = Student(strategie1)
student2 = Student(strategie2)

student2.ascultaGluma(student1.spuneGluma())
student1.ascultaGluma(student2.spuneGluma())
```

threading_intreteserea_linieCUlinie

```
import threading
from functional import seq

#intretesare linii din fisiere in adt
#prelucrare linii din adt

class Tailor(threading.Thread):
def __init__(self,file1,file2,file3,file4):
threading.Thread.__init__(self)
self.file1=file1
self.file2=file2
self.file3=file3
self.file4=file4
self.lines=[]
def sew_lines(self):
with open(self.file1) as fis1:
with open(self.file2) as fis2:
with open(self.file3) as fis3:
with open(self.file4) as fis4:

carry_on=True
while carry_on:
carry_on=False

line1=fis1.readline()
if len(line1) >0:
self.lines.append(line1)
carry_on=True

for _ in range(0,2):
line2=fis2.readline()
if len(line2) >0:
self.lines.append(line2)
carry_on=True

for _ in range(0,3):
line3=fis3.readline()
if len(line3) >0:
self.lines.append(line3)
carry_on=True

for _ in range(0,4):
line4=fis4.readline()
if len(line4) >0:
self.lines.append(line4)
carry_on=True

self.lines=seq(self.lines).map(func=lambda line: line.strip())

with open("result.txt","w") as result:
for line in self.lines:
result.write(line)

def run(self):
self.sew_lines()

def main():
tailor=Tailor("fis1.txt","fis2.txt","fis3.txt","fis4.txt")
tailor.start()
main()
#simulare
```

tkinter_lambda_mouse

```
from tkinter import *
from queue import LifoQueue

class Drawer(Frame):
def __init__(self, root, **kw):
super().__init__(**kw)
self.root = root
self.stack = LifoQueue()
self.canvas = Canvas(root, width=900, height=900)
self.canvas.grid(row=1, column=0)
self.canvas.bind('<Button-1>', self.addPoint)
self.label = Label(self.canvas, text="Label lol")
self.label.place(relx=0.5, rely=0.05, anchor=CENTER)
self.buttonQuit = Button(self.root, text="Quit", padx=20, pady=20, command=self.quit).grid(row=0, column=0)
self.buttonClear = Button(self.root, text="Clear", padx=20, pady=20, command=self.clear).grid(row=0, column=1)
#self.canvas.pack()
def addPoint(self, event):
```

```
self.label.configure(text="Am pus un punct la " + str(event.x) + " si " + str(event.y))
if not self.stack.empty():
point1 = self.stack.get()
self.canvas.create_line(point1[0], point1[1], event.x, event.y)
#self.canvas.pack()
self.stack.put((event.x, event.y))
def quit(self):
import sys
sys.exit()
def clear(self):
self.label.configure(text="Am sters tot!")
self.canvas.delete("all")
#self.canvas.pack()
while not self.stack.empty():
self.stack.get()

if __name__ == '__main__':
root = Tk()
root.title("cod maimuta")
drawer = Drawer(root)
root.mainloop()
sv38py346
```

KOTLIN

2CARACTERE_DIN_MIJLOC_CUVANT_STERGERE

```
import java.io.File

fun main(args:Array<String>) {
var a=""
File("in.txt").forEachLine { a+=it
a+=" "
}
val words:List<String>
words = a.split(" ")

val new_words=words.filter { it.length >= 4 }
.map { it.removeRange(it.length/2-1,it.length/2+1)}

println(new_words)

}

///Stergere sv34 Kot40
import java.io.File

fun main(args:Array<String>) {
var a=""
File("out.txt").forEachLine { a+=it
a+=" "
}
val words:List<String>
words = a.split(" ")

val new_words=words.filter { it.length >= 4 }
.map { it.removeRange(0,2)}

println(new_words)

}
```

2colectii,15nr,aleator,AxB

```
fun main(args:Array<String>) {

val A = mutableListOf<Int>()
val B = mutableListOf<Int>()

for (i in 0..15) {
A.add((0..100).random())
B.add((0..100).random())
}

println(A)
println(B)

var AxB = A.map { x->B.map { y->Pair(x,y) } }
println(AxB)
//sv44 kot57


```

2_colectii_100nr_functie_multimi

```
val compunere={
A:Set<Int>,B:Set<Int> ->
var X = setOf<Pair<Int,Int>>()
for (i in A){
for (j in B)
{
var p=Pair(i,j)
X+=p
}
}
X
}
val intersectie={
A:Set<Int>,B:Set<Int> ->
var X = setOf<Int>()
for (i in A){
for (j in B)
{
if(i==j)
X+=i
}
}
X
}
val reuniune={
A:Set<Pair<Int,Int>>,B:Set<Int> ->
var X = hashMapOf<Int,Int>()
for (i in A){
X+=i
}
for(j in B)
{
X+=Pair(j,0)
}
X
}
fun main(){
var A = mutableSetOf<Int>()
var B = mutableSetOf<Int>()
var countA=0
var countB=0
var i=0
while(countA<100)
{
var numarator=8*i-18
var numitor=2*i-9
if(numarator/numitor >=0)
{
A+=numarator/numitor
countA+=1
}
i+=1
println(countA)
}
i=0
while(countB<100)
{
var numarator=9*i*i -48*i +16
var numitor=3*i-8
if(numarator/numitor >=0)
{
```

```
B+=numarator/numitor
countB+=1
}
i+=1
println(countB)
}
println("A="+A)
println("B="+B)
/*var AxB=compunere(A,B)
println("(AxB)="+AxB)
var BxA=intersectie(B,A)
println("(BnA)="+BxA)
var comun=reuniune(AxB,BxA)
println("(AxB) u (BnA)="+comun)*/

var AxB = A.map { x->B.map { y->Pair(x,y) } }
// println(AxB)

var AXB= mutableListOf<Pair<Int, Int>>()
AxB.forEach { it.forEach { AXB.add(it) } }
println(AXB)

var BnA= B.filter { A.contains(it) }
println(BnA)

var result = AXB+BnA
println(result)
}
//sv53 kot65


```

2_colectii_100nr_functie_multimi #varianta2

```
fun main(args:Array<String>) {

val A = mutableListOf<Int>()
val B = mutableListOf<Int>()

for (i in 0..99)
A.add((8*i-18)/(2*i-9))
for (i in 0..99)
B.add((9*i*i-48*i+16)/(3*i-8))

println(A)
println(B)

var AxB = A.map { x->B.map { y->Pair(x,y) } }
// println(AxB)

var AXB= mutableListOf<Pair<Int, Int>>()
AxB.forEach { it.forEach { AXB.add(it) } }
println(AXB)

var BnA= B.filter { A.contains(it) }
println(BnA)

var result = AXB+BnA
println(result)


////SAU
import kotlin.math.floor

fun generatecollection1(): MutableList<Float> {


val coll = mutableListOf<Float>()

var index = 0
var n: Int = 0
var x: Float

while (index < 100 && n < 1000)
{
x = ((8.0 * n - 18.0) / (2.0 * n - 9.0)).toFloat()
if(floor(x) == x && x > 0) {
coll.add(x)
//println("chestie gasita [$index] [$n] [$x]")
index++
}
n++
}
return coll
}


fun generatecollection2():MutableList<Float>
{
```

```
val coll = mutableListOf<Float>()

var index = 0
var n: Int = 0
var x: Float

while (index < 100 && n < 1000)
{
x = ((9.0 * n * n - 48.0 * n + 16.0) / (3.0 * n - 8.0)).toFloat()
if(floor(x) == x ) {

coll.add(x)
//println("chestie gasita [$index] [$n] [$x]")
index++

}
n++
}
return coll
}

fun main() {
var collection1:List<Float> = generatecollection1()
collection1 = collection1.sorted()
println("Colectia 1: $collection1")

var collection2:List<Float> = generatecollection2()
collection2 = collection2.sorted()
println("Colectia 2: $collection2")

var hashmap = (1..listOf(collection1.map { c1-> collection2.map { c2-> Pair(c1,c2)}}.flatten(),
collection1.intersect(collection2)).flatten().size).zip(
listOf(collection1.map { c1-> collection2.map { c2-> Pair(c1,c2)}}.flatten(), collection1.intersect(collection2)).flatten()).toMap()

print(hashmap)

}
sv53 kot65

```

AB_AxB_lambda

```
import java.util.Random

val compunere={
A:Set<Int>,B:Set<Int> ->
var X = setOf<Pair<Int,Int>>()
for (i in A){
for (j in B)
{
var p=Pair(i,j)
X+=p
}
}
X
}

fun main()
{
var A = setOf<Int>()
var B = setOf<Int>()
for (i in 1..16)
{
A+= (0..999).random()
B+= (0..999).random()
}
var X=compunere(A,B)
println("AxB="+X)

var AxB = A.map { x->B.map { y->Pair(x,y) } }
println(AxB)
}
//sv44kot57

```

---

adaptor_datecolectii_afisareca_operatie

```
import java.io.File

interface NUMBER
{
fun afiseazaDouble(fw:File)
fun afiseazaFloat(fw:File)
fun afiseazaInt(fw:File)
}
class DOUBLE(val nr: Number):NUMBER
```

```
{
override fun afiseazaDouble(fw:File) {
fw.appendText("Am afisat un double: " + nr + "\n")
}
override fun afiseazaFloat(fw:File) {
}
override fun afiseazaInt(fw:File) {
}
override fun toString(): String {
return "double"
}
}
class INTEGER(val nr: Number):NUMBER
{
override fun afiseazaDouble(fw:File) {
}
override fun afiseazaFloat(fw:File) {
}
override fun afiseazaInt(fw:File) {
fw.appendText("Am afisat un intreg: " + nr + "\n")
}
override fun toString(): String {
return "int"
}
}

class FLOAT(val nr: Number):NUMBER
{
override fun afiseazaDouble(fw:File) {
}

override fun afiseazaFloat(fw:File) {
fw.appendText("Am afisat un float: " + nr + "\n")
}

override fun afiseazaInt(fw:File) {

}
override fun toString(): String {
return "float"
}
}

class Complex(val re:Number, val im:Number)
{
fun afiseazaComplex(fw: File)
{
fw.appendText("Am afisat un complex: " + re +" " +im+"*i" + "\n")
}
override fun toString(): String {
return "complex"
}
}
interface FileWriter
{
fun Write(fw: File, obj: Any)
}

open class Adapter:FileWriter
{
override fun Write(fw:File, obj: Any) {
val aux: NUMBER = obj as NUMBER
when(aux.toString())
{
"double"-> aux.afiseazaDouble(fw)
"int"-> aux.afiseazaInt(fw)
"float"-> aux.afiseazaFloat(fw)
else -> fw.appendText("numar incorect" + "\n")
}
}
}

class Executor:Adapter()
{
override fun Write(fw: File, obj: Any) {
if (obj.toString() == "double" || obj.toString() == "int" || obj.toString() == "float")
{
super.Write(fw, obj)
}
else
if(obj.toString() =="complex")
{
val aux: Complex = obj as Complex
aux.afiseazaComplex(fw)
}
}
}
fun main(args:Array<String>)
{
val file = File("out.txt")
val executor = Executor()

```

```
val intreg = INTEGER(700)
val real = DOUBLE(800.6)
val floatNr = FLOAT(32.6f)
val complex = Complex(10,80)

executor.Write(file,intreg)
executor.Write(file,real)
executor.Write(file,floatNr)
executor.Write(file,complex)
}
//sv81 kot71
```

---

adaptor_datecolectii_afisare_ca_o_singura_operatie

```
import java.io.File

interface NUMBER
{
fun afiseazaDouble()
fun afiseazaFloat()
fun afiseazaInt()
}
class DOUBLE(val nr: Number):NUMBER
{
override fun afiseazaDouble() {
println("Am afisat un double: " + nr + "\n")
}
override fun afiseazaFloat() {
}
override fun afiseazaInt() {
}
override fun toString(): String {
return "double"
}
}
class INTEGER(val nr: Number):NUMBER
{
override fun afiseazaDouble() {
}
override fun afiseazaFloat() {
}
override fun afiseazaInt() {
println("Am afisat un intreg: " + nr + "\n")
}
override fun toString(): String {
return "int"
}
}

class FLOAT(val nr: Number):NUMBER
{
override fun afiseazaDouble() {
}

override fun afiseazaFloat() {
println("Am afisat un float: " + nr + "\n")
}

override fun afiseazaInt() {

}
override fun toString(): String {
return "float"
}
}

class Complex(val re:Number, val im:Number)
{
fun afiseazaComplex()
{
println("Am afisat un complex: " + re +" " +im+"*i" + "\n")
}
override fun toString(): String {
return "complex"
}
}
interface Writer
{
fun Write(obj: Any)
}

open class Adapter:Writer
{
override fun Write(obj: Any) {
val aux: NUMBER = obj as NUMBER
when(aux.toString())
```

```
{
"double"-> aux.afiseazaDouble()
"int"-> aux.afiseazaInt()
"float"-> aux.afiseazaFloat()
else -> println("numar incorect" + "\n")
}
}
}

class Executor:Adapter()
{
override fun Write(obj: Any) {
if (obj.toString() == "double" || obj.toString() == "int" || obj.toString() == "float")
{
super.Write(obj)
}
else
if(obj.toString() =="complex")
{
val aux: Complex = obj as Complex
aux.afiseazaComplex()
}
}
}
fun main(args:Array<String>) {
val file = File("out.txt")
val executor = Executor()

val intreg = INTEGER(700)
val real = DOUBLE(800.6)
val floatNr = FLOAT(32.6f)
val complex = Complex(10, 80)

executor.Write(intreg)
executor.Write(real)
executor.Write(floatNr)
executor.Write(complex)
}
//sv80 kot70
```

---

adt_lambda_nr_submultimi_4el

```
val procesareSubmultimi={
A:Set<Int> ->
var array:MutableSet<MutableSet<Int>> = mutableSetOf()
subset(array, mutableSetOf<Int>(),A,0)
array
}
fun subset(S: MutableSet<MutableSet<Int>> , resultSet:MutableSet<Int>, A: Set<Int>,start: Int)
{
S.add(mutableSetOf<Int>())
for(i in start..A.size-1)
{
resultSet.add(A.elementAt(i))
subset(S,resultSet,A,i+1)
resultSet.remove(resultSet.size - 1)
}

}

fun main()
{
val A=1.until(101).toSet()
var count = 0
val subsets=procesareSubmultimi(A)
print(subsets)
subsets.forEach {
if(it.size == 4 && it.contains(1))
count+=1
}
println("Numarul submultimilor cu 4 elemente si care contin elementul 1:"+count)
}
//sv38kot52
```

---

arbore_binar_functor_gasire_parsefile

```
import java.io.File

class ArboreBinar(var cuvant: String) {
var left: ArboreBinar? = null
var right: ArboreBinar? = null
}
```

```
2123.
2124. class FunctorClass(val rad: ArboreBinar?){
2125. fun map(function1: ((ArboreBinar) -> ArboreBinar?)? = null, function2: ((ArboreBinar) -> ArboreBinar?)? = null): Unit {
2126. if (rad != null) {
2127. println(rad.cuvant + " ")//afisam tot arborele
2128. FunctorClass(function1?.invoke(rad)).map(function1,function2)//trecem la stanga
2129. FunctorClass(function2?.invoke(rad)).map(function1,function2)//trecem la dreapta
2130. }
2131. }
2132. }
2133.
2134. fun nextLeft(rad: ArboreBinar) : ArboreBinar?{
2135. //if(rad.left != null )print("pe stanga: ")
2136. return rad.left
2137. }
2138.
2139. fun nextRight(rad: ArboreBinar) : ArboreBinar?{
2140. //if(rad.right != null )print("pe dreapta: ")
2141. return rad.right
2142. }
2143.
2144. fun findInTree(rad: ArboreBinar, cuv: String) : ArboreBinar?{
2145. var rez : ArboreBinar? = null
2146. if(rad.cuvant == cuv)
2147. return rad
2148. if(cuv.compareTo(rad.cuvant) < 0 || (cuv.compareTo(rad.cuvant) == 0 && cuv.length < rad.cuvant.length)) // alfabetic, la egalitate
     dupa dimensiune?
2149. {
2150. rez = findInTree(rad.left!!, cuv)
2151. }
2152. else {
2153. rez = findInTree(rad.right!!, cuv)
2154. }
2155. return rez
2156. }
2157.
2158. fun parseFile(path: String) : List<String>
2159. {
2160. val lista = mutableListOf<String>()
2161. File(path).forEachLine {
2162. lista.addAll(it.split(" "))
2163. }
2164. return lista
2165. }
2166.
2167. fun makeTree(lista: List<String>) : ArboreBinar
2168. {
2169. val rad = ArboreBinar(lista[0])
2170. val list = lista.drop(1)//tot fara primul cuvant din string
2171. list.forEach{
2172. insertTree(rad,it)
2173. }
2174. return rad
2175. }
2176.
2177. fun insertTree(rad: ArboreBinar, cuv:String){
2178. if(cuv < rad.cuvant) // alfabetic
2179. {
2180. if(rad.left == null)
2181. rad.left = ArboreBinar(cuv)
2182. else
2183. insertTree(rad.left!!,cuv)
2184. }
2185. else
2186. {
2187. if(rad.right == null)
2188. rad.right = ArboreBinar(cuv)
2189. else
2190. insertTree(rad.right!!,cuv)
2191. }
2192. }
2193.
2194. fun main(args: Array<String>) {
2195. val name = "Fisier.txt"
2196. val arbore = makeTree(parseFile(name))
2197. println("\nAfisare tot arborele:")
2198. FunctorClass(arbore).map(::nextLeft, ::nextRight) // afiseaza tot arborele
2199. println("\nAfisare tot subarborele cu radacina in \"monke\"")
2200. FunctorClass(findInTree(arbore, "monke")).map(::nextLeft, ::nextRight) // afiseaza tot subarborele cu radacina in "monke"
2201. println("\nAfisare tot subarborele stanga cu radacina in \"monke\"")
2202. FunctorClass(findInTree(arbore, "monke")).map(::nextLeft) // afiseaza tot subarborele stang cu radacina in "monke"
2203. println("\nAfisare tot subarborele drept cu radacina in \"monke\"")
2204. FunctorClass(findInTree(arbore, "monke")).map(::nextRight) // afiseaza tot subarborele drept cu radacina in "monke"
2205.
2206. }
```

AuB x BintersectA_dictionar

```
val compunere={
A:Set<Int>,B:Set<Int> ->
var X = hashMapOf<Int,Int>()
for (i in A){
for (j in B)
{
var p=Pair(i,j)
if(!X.containsKey(i) || !X.containsValue(j))
X+=p
}
}
X
}
val intersectie={
A:Set<Int>,B:Set<Int> ->
var X = setOf<Int>()
for (i in A){
for (j in B)
{
if(i==j)
X+=i
}
}
X
}
val reuniune={
A:Set<Int>,B:Set<Int> ->
var X = setOf<Int>()
for (i in A){
X+=i
}
for(j in B)
{
if(!X.contains(j))
X+=j
}
X
}
fun main()
{
var A = mutableSetOf<Int>()
var B = mutableSetOf<Int>()
for (i in 1..21)
{
A+= (0..5).random()
B+= (0..5).random()
}
println("A="+A)
println("B="+B)
/*var AuB=reuniune(A,B)
println("(AuB)="+AuB)
var BnA=intersectie(B,A)
println("(BnA)="+BnA)
var comun=compunere(AuB,BnA)
println("(AuB) x (BnA)="+comun)*/

val AuB = A.filter { !B.contains(it) } + B
println(AuB)

val AnB =A.filter { B.contains(it) }
println(AnB)

var AuBxBnA = AuB.map { x->AnB.map { y->Pair(x,y) } }
println(AuBxBnA)

var AUBXBNA= mutableListOf<Pair<Int, Int>>()
AuBxBnA.forEach { it.forEach { AUBXBNA.add(it) } }
println(AUBXBNA.toMap())
}

SAU
///
fun main(args:Array<String>) {

val A = mutableListOf<Int>()
val B = mutableListOf<Int>()

for (i in 0..5)
A.add(i)
for (i in 3..8)
B.add(i)

println(A)
println(B)

```

```
val AuB = A.filter { !B.contains(it) } + B
println(AuB)

val AnB =A.filter { B.contains(it) }
println(AnB)

var AuBxBnA = AuB.map { x->AnB.map { y->Pair(x,y) } }
println(AuBxBnA)

var AUBXBNA= mutableListOf<Pair<Int, Int>>()
AuBxBnA.forEach { it.forEach { AUBXBNA.add(it) } }
println(AUBXBNA.toMap())

sv50 kot62
```

---

automat_inchidere_paranteze

```
import java.io.BufferedReader
import java.io.File

class Automat
{
var state: State
var par_deschise: Int = 0
init{
state = ParantezeState(this, "C:\\Users\\monic\\OneDrive\\Desktop\\kotlinpp\\kotlinSource.txt")
}
fun verificaInchidereCorecta(s: String): Boolean = state.verificaInchidereCorecta(s)
fun extrageParanteze(): String = state.extrageParanteze()
}

abstract class State(val automat: Automat, val filename: String)
{
abstract fun extrageParanteze(): String

abstract fun verificaInchidereCorecta(s: String): Boolean

fun eParantezaDeschisa(c: Char): Boolean
{
if(c =='(' || c == '[' || c == '{')
return true
return false
}

fun eParantezaInchisa(c: Char): Boolean
{
if(c ==')' || c == ']' || c == '}')
return true
return false
}
}

class ParantezeState(automat: Automat, filename: String): State(automat,filename)
{
override fun extrageParanteze(): String
{
var result :String = ""
val bufferedReader: BufferedReader = File(filename).bufferedReader()
val inputString = bufferedReader.use { it.readText() }
var i = 0
while(i < inputString.length)
{
if(eParantezaDeschisa(inputString[i])) {
automat.par_deschise += 1
result += inputString[i]
}
if (eParantezaInchisa(inputString[i])) {
automat.par_deschise -= 1
result += inputString[i]
}
i += 1
}
return result
}

override fun verificaInchidereCorecta(s: String): Boolean
{
if(automat.par_deschise == 0)
return true
return false
}
}


fun main(args: Array<String>)
{
val automat = Automat()
var paranteze = automat.extrageParanteze()
```

```
println(paranteze)

var verifica = automat.verificaInchidereCorecta(paranteze)

if(verifica == true)
println("Parantezele au fost inchise corect!")
else
println("Parantezele nu au fost inchise corect!")
}
```

AxB U BxB_dictionar

```
val compunere={
A:Set<Int>,B:Set<Int> ->
var X = setOf<Pair<Int,Int>>()
for (i in A){
for (j in B)
{
var p=Pair(i,j)
X+=p
}
}
X
}
val reuniune={
A:Set<Pair<Int,Int>>,B:Set<Pair<Int,Int>> ->
var X = hashMapOf<Int,Int>()
for (i in A){
X+=i
}
for(j in B)
{
if(!X.containsKey(j.first) || !X.containsValue(j.second))
X+=j
}
X
}
fun main()
{
var A = mutableSetOf<Int>()
var B = mutableSetOf<Int>()
for (i in 1..21)
{
A+= (0..5).random()
B+= (0..5).random()
}
println("A="+A)
println("B="+B)
/*var AxB=compunere(A,B)
println("(AxB)="+AxB)
var BxB=compunere(B,B)
println("(BxA)="+BxB)
var comun=reuniune(AxB,BxB)
println("(AxB) u (BxB)="+comun)*/

var AxB = A.map { x->B.map { y->Pair(x,y) } }
// println(AxB)

var AXB= mutableListOf<Pair<Int, Int>>()
AxB.forEach { it.forEach { AXB.add(it) } }
println(AXB)

var BxB = B.map { x->B.map { y->Pair(x,y) } }
//println(BxB)

var BXB= mutableListOf<Pair<Int, Int>>()
BxB.forEach { it.forEach { BXB.add(it) } }
println(BXB)

var AxBuBxB= AXB.filter { !BXB.contains(it) }
AxBuBxB+=(BXB.filter { BXB.contains(it) })
println(AxBuBxB)

println(AxBuBxB.toMap())
}
SAU
////////////

fun main(args:Array<String>) {

val A = mutableListOf<Int>()
val B = mutableListOf<Int>()

for (i in 0..2)
A.add(i)
for (i in 1..3)
B.add(i)
```

```
println(A)
println(B)

var AxB = A.map { x->B.map { y->Pair(x,y) } }
println(AxB)

var BxA = B.map { x->A.map { y->Pair(x,y) } }
println(BxA)

var AXB= mutableListOf<Pair<Int, Int>>()
AxB.forEach { it.forEach { AXB.add(it) } }

var BXA= mutableListOf<Pair<Int, Int>>()
BxA.forEach { it.forEach { BXA.add(it) } }

println(AXB)
println(BXA)

val result = AXB.filter { BXA.contains(it) }.toMap()
println(result)

}

//sv47 kot60
```

AxB x BuA_dictionar

```
fun main(args:Array<String>) {

val A = mutableListOf<Int>()
val B = mutableListOf<Int>()

for (i in 0..2)
A.add(i)
for (i in 1..3)
B.add(i)

println(A)
println(B)

var AxB = A.map { x->B.map { y->Pair(x,y) } }
println(AxB)

var AXB= mutableListOf<Pair<Int, Int>>()
AxB.forEach { it.forEach { AXB.add(it) } }
println(AXB)

var BuA= A.filter { !B.contains(it) }
BuA+=(B.filter { B.contains(it) })
println(BuA)

var AxBxBuA = AXB.map { x->B.map { y->Pair(x,y) } }
println(AxBxBuA)

var AXBXBUA= mutableListOf<Pair<Pair<Int, Int>,Int>>()
AxBxBuA.forEach { it.forEach { AXBXBUA.add(it) } }

println(AXBXBUA.toMap())


}

//////////////sau
val reuniune={
A:Set<Int>,B:Set<Int> ->
var X = setOf<Int>()
for (i in A){
X+=i
}
for(j in B)
{
if(!X.contains(j))
X+=j
}
X
}
val compunere={
A:Set<Int>,B:Set<Int> ->
var X = setOf<Pair<Int,Int>>()
for (i in A){
for (j in B)
{
var p=Pair(i,j)
X+=p
}
```

```
}
X
}
val compuneref={
A:Set<Pair<Int,Int>>,B:Set<Int> ->
var X = hashMapOf<Pair<Int,Int>,Int>()
for (i in A){
for (j in B)
{
var p=Pair(i,j)
if(!X.containsKey(i) || !X.containsValue(j))
X+=p
}
}
X
}
fun main()
{
var A = mutableSetOf<Int>()
var B = mutableSetOf<Int>()
for (i in 0..2)
A.add(i)
for (i in 1..3)
B.add(i)
println("A="+A)
println("B="+B)
/*var AxB=compunere(A,B)
println("(AxB)="+AxB)
var BxB=reuniune(B,A)
println("(BuA)="+BxB)
var comun=compuneref(AxB,BxB)
println("(AxB) x (BuA)="+comun)*/

var AxB = A.map { x->B.map { y->Pair(x,y) } }
println(AxB)

var AXB= mutableListOf<Pair<Int, Int>>()
AxB.forEach { it.forEach { AXB.add(it) } }
println(AXB)

var BuA= A.filter { !B.contains(it) }
BuA+=(B.filter { B.contains(it) })
println(BuA)

var AxBxBuA = AXB.map { x->B.map { y->Pair(x,y) } }
println(AxBxBuA)

var AXBXBUA= mutableListOf<Pair<Pair<Int, Int>,Int>>()
AxBxBuA.forEach { it.forEach { AXBXBUA.add(it) } }

println(AXBXBUA.toMap())
}
//sv51Kot63
```

AxBintersectBxA_dictionar

```
val compunere={
A:Set<Int>,B:Set<Int> ->
var X = setOf<Pair<Int,Int>>()
for (i in A){
for (j in B)
{
var p=Pair(i,j)
X+=p
}
}
X
}
val intersectie={
A:Set<Pair<Int,Int>>,B:Set<Pair<Int,Int>> ->
var X = hashMapOf<Int,Int>()
for (i in A){
for (j in B)
{
if(i.first==j.first && i.second==j.second)
if(!X.containsKey(i.first) || !X.containsValue(i.second))
X.put(i.first,i.second)
}
}
X
}
fun main()
{
var A = mutableSetOf<Int>()
var B = mutableSetOf<Int>()
for (i in 1..21)
{
```

```
A+= (0..5).random()
B+= (0..5).random()
}
println("A="+A)
println("B="+B)
/*var AxB=compunere(A,B)
println("(AxB)="+AxB)
var BxA=compunere(B,A)
println("(BxA)="+BxA)
var comun=intersectie(AxB,BxA)
println("(AxB) n (BxA)="+comun)*/

var AxB = A.map { x->B.map { y->Pair(x,y) } }
println(AxB)

var BxA = B.map { x->A.map { y->Pair(x,y) } }
println(BxA)

var AXB= mutableListOf<Pair<Int, Int>>()
AxB.forEach { it.forEach { AXB.add(it) } }

var BXA= mutableListOf<Pair<Int, Int>>()
BxA.forEach { it.forEach { BXA.add(it) } }

println(AXB)
println(BXA)

val result = AXB.filter { BXA.contains(it) }.toMap()
println(result)
}

//////////
//SAU
fun main(args:Array<String>) {

val A = mutableListOf<Int>()
val B = mutableListOf<Int>()

for (i in 0..2)
A.add(i)
for (i in 1..3)
B.add(i)

println(A)
println(B)

var AxB = A.map { x->B.map { y->Pair(x,y) } }
println(AxB)

var BxA = B.map { x->A.map { y->Pair(x,y) } }
println(BxA)

var AXB= mutableListOf<Pair<Int, Int>>()
AxB.forEach { it.forEach { AXB.add(it) } }

var BXA= mutableListOf<Pair<Int, Int>>()
BxA.forEach { it.forEach { BXA.add(it) } }

println(AXB)
println(BXA)

val result = AXB.filter { BXA.contains(it) }.toMap()
println(result)

}

//sv46kot59
------------------------------------------------------------------------
corutine_text_kotlin
------------------------------------------------------------------
import kotlinx.coroutines.*
import kotlinx.coroutines.channels.Channel
import kotlinx.coroutines.sync.Mutex
import kotlinx.coroutines.sync.withLock
import java.io.File

fun main() = runBlocking <Unit>{
val mutex= Mutex()
val channel = Channel<String>(Channel.CONFLATED)
var final : Boolean = false
var which : Int = 1
launch {
var s :String
var s1 : String
val f = File("/home/mihai/git-
    server/repos/1211a_zaharia_teodor_mihai.git/Laborator10/Laborator10/src/main/kotlin/Pregatire/Text.txt").bufferedReader()
while(f.ready()) {
s = f.readLine()
while(s.indexOf(" ")!=-1)
{
s1=s.substring(0,s.indexOf(" "))
if(channel.isEmpty) {
```

```
channel.send(s1)
s = s.substring(s.indexOf(" ") + 1, s.length)
}
else
yield()
}
yield()
channel.send(s)
}
final = true
}
launch {
while (!channel.isEmpty||final==false)
if(which==1) {
stringReceive(1, channel)
which=2
}
else{
yield()
}
}
launch {
while (!channel.isEmpty||final==false)
if(which==2) {
stringReceive(2, channel)
which=3
}
else{
yield()
}
}
launch {
while (!channel.isEmpty||final==false)
if(which==3) {
stringReceive(3, channel)
which=1
}
else{
yield()
}
}
}
suspend fun stringReceive(i:Int, c : Channel<String>)
{
var s: String = "";
s=c.receive()
println("Mesajul "+s+" transmis de corutina "+i)
}
```

---

funct_colectii_sumCuMult

```
object functie{
fun <A, B> pura(b: B) = {_:A ->b}
}
fun <A,B,C> ((A) -> B).map(transform:(B) -> C): (A) -> C = {t -> transform(this(t))}
fun <A,B,C> ((A) -> B).flatMap(fm:(B) -> (A) -> C): (A) -> C = {t -> fm(this(t))(t)}
fun <A,B,C> ((A) -> B).ap(fab:(A)->(B) -> C): (A) -> C = fab.flatMap{f -> map(f)}

fun main()
{
val sumCu5SiMulCu7: (Int) -> Int = {i: Int -> i+5}.ap{{j:Int -> j*7}}
println(sumCu5SiMulCu7(7))
println(sumCu5SiMulCu7(8))
println(sumCu5SiMulCu7(9))
}
```

---

generice_functiiextensie_corutina_lambda_list

```
import kotlinx.coroutines.*
import java.io.File
import java.util.*

// suspectez ca functia de extensie era pe domeniul corutinelor
suspend fun CoroutineScope.transform(list: List<Int>): List<Int> =
list.asSequence()
.map{it -> it/3}
.map{it -> it*5}
.toList()

fun main()= runBlocking<Unit> {
//ain't nobody got time for that
```

```
//val scanner=Scanner(System.`in`)
//println("Filename expected:")
val fileName="numbers.txt"//scanner.nextLine()

val file=File(fileName)
val lines= file.inputStream().bufferedReader().readText()

val myList= lines.split(regex = Regex("\\W"))
.asSequence()
.map{it->it.toInt()}
.toList()
val res= async{transform(myList)}
print(res.await())

}
```

---

hasmap_functor_lambda

```
val procesareLambda={x:Int -> 3*x-1}
fun HashMap<Int, Int>.converttoStr() : HashMap<String, String> =
if(size>0){
val MapNou = HashMap<String, String>(size)
for(element in this){
MapNou.put(element.key.toString(),procesareLambda(element.key).toString())
}
MapNou
}else{
val MapEmpty = HashMap<String, String>()
MapEmpty//nu exista emptyHashMap()
}
fun main(){
val HM= hashMapOf(
3 to procesareLambda(3),
5 to procesareLambda(5),
6 to procesareLambda(6), //functia si functorul va lucra numai cu cheile din HashMap-ul acesta
7 to procesareLambda(7)
)
println(HM.converttoStr())
}
Alternativ, cred
fun main(args:Array<String>) {
var my_map = hashMapOf<Int, Int>()
for (i in 0..100)
{
my_map[i] = i
}
println(my_map)
my_map.values.map { i-> 3 * i - 1 }
.map(Int::toString)
.forEach ( ::println )

}
```

---

memoizarea_calculsir_hasmap

```
import java.util.concurrent.ConcurrentHashMap

fun <A,R> Function1<A,R>.memorized():(A)->R{
val map=ConcurrentHashMap<A,R>()
return {a->map.getOrPut(a){this.invoke(a)} }
}
fun f(n:Int):Int{
if (n<1)
return 0
else if(n==1)
return 1
else
return f(n-1)+f(n-2)
}
fun main(){
val memoizedFunction={i:Int -> f(i)}.memorized()//memorizare generalizata
print("n=")
val n= readLine()!!.toInt()
for(i in 1..n) {
print("\nf(${i})=")
print(memoizedFunction(i))
}
}
sv108 kot101
```

---

oop_oameni_beau_danseaza

```
import java.util.*

class Human(name:String, food:String,drink:String, interest:String)
{
var name = name;
var food = food;
var drink = drink;
var interest = interest;
fun display()
{
println("name: " + name + " food: "+ food +" drink: "+ drink + " interest: " + interest )
}
}
fun Question(text:String): List<String>
{
val list_of_words = text.split("\\W+".toRegex())
return list_of_words
}
fun DataMiner(humans:List<Human>, sentence: List<String>):Boolean {
var human_found = false
var food_negation_found = false
var food_found = false
var drink_negation_found = false
var drink_found = false
var interest_negation_found = false
var interest_found = false
var b=false
for (human in humans) {
human_found = false
food_negation_found = false
food_found = false
var drink_negation_found = false
var drink_found = false
interest_negation_found = false
interest_found = false

for (element in sentence) {
//numele
if (human.name.equals(element) || human_found) {
human_found = true
}
//mancare
if(human_found && element.equals("nu")) {
food_negation_found = true
}

if (human.food.equals(element)) {
food_found = true
}
//bautura
if(food_found && element.equals("nu")) {
drink_negation_found = true
}

if (human.drink.equals(element)) {
drink_found = true
}
//placeri
if(drink_found && element.equals("nu")) {
interest_negation_found = true
}

if (human.interest.equals(element)) {
interest_found = true
}

if(drink_negation_found && food_negation_found && interest_negation_found && human_found && !food_found && !drink_found &&
    !interest_found) {
b= true
}
else
{
if (human_found && interest_found && drink_found && food_found && !food_negation_found && !drink_negation_found &&
    !interest_negation_found)
b= true
}
}
}
return b
}
fun main() {

val human1 = Human("Alex", "pizza", "vin","sefii")
val human2 = Human("Vasile", "pere", "vodka","barbati")
val human3 = Human("Ion", "mere","bere" ,"femei")
var humans: List<Human> = listOf(human1, human2, human3)
```

```
//afisam omuleti
for (element in humans) {
element.display()
}

val read = Scanner(System.`in`)

println("Enter String:")
val text = readLine()
if(text == null){
println("error")
} else {
val word = Question(text)
val trust = DataMiner(humans, word)
println(trust)
}
}
//sv58 kot86
```

---

oop_salacurs_enum

```
enum class Randuri{
RAND1,RAND2,RAND3,RAND4,RAND5,RAND6
}
enum class Geamuri{
GeamStanga,GeamDreapta,GeamFata,GeamSpate
}
class Proiector(val id: Int)
{
fun Open()
{
println("Profesorul a deschis proiectorul ${id}")
}
}
class Elev(val nume: String)
{
fun SePregatesteDeTest(rand:Randuri)
{
println("Elevul " + nume + " se aseaza pe randul " + rand)
}
}
class Profesor(val nume:String)
{
fun DeschideProiectorul(p:Proiector)
{
p.Open()
}

fun AseazaEleviTest(elevi: MutableList<Elev>)
{
for (i in 0..elevi.size-1)
{
elevi[i].SePregatesteDeTest(Randuri.values()[i])
}
}

fun InchideGeamnul(directie: String)
{
when(directie)
{
"stanga"->{
println("Profesorul a deschis: " + Geamuri.GeamStanga)
}
"dreapta"->{
println("Profesorul a deschis: " + Geamuri.GeamDreapta)
}
"fata"->{
println("Profesorul a deschis: " + Geamuri.GeamFata)
}
"spate"->{
println("Profesorul a deschis: " + Geamuri.GeamSpate)
}
else -> println("Nu exista geam acolo")
}
}
}

class SalaDeCurs(val profesor:Profesor, val elevi: MutableList<Elev>, val proiector:Proiector)
{
fun ExecuteSomething(actiune: String)
{
when(actiune) {
"proiector" -> {
profesor.DeschideProiectorul(proiector)
}
"geam"->{
```

```
profesor.InchideGeamnul("stanga")
}
"test"->{
profesor.AseazaEleviTest(elevi)
}
else->{
println("Actiunea nu a fost dezvoltata inca")
}
}
}
}
fun main(args:Array<String>) {

val elev1 = Elev("Alex")
val elev2 = Elev("Marian")
val elev3 = Elev("George")
val elev4 = Elev("Tudor")
val elev5 = Elev("Hara")
val elev6 = Elev("Ionut")
val elevi = mutableListOf<Elev>(elev1,elev2,elev3,elev4,elev5,elev6)

val proiector = Proiector(1)
val profesor = Profesor("Mihai")

val salaDeCurs = SalaDeCurs(profesor,elevi,proiector)

salaDeCurs.ExecuteSomething("proiector")
salaDeCurs.ExecuteSomething("geam")
salaDeCurs.ExecuteSomething("test")
}
//sv65 kot95
```

---

oop_salaLab_listeMutable

```
open class Om(val nume:String){
fun Scrie_Tabla(){
println(nume + " scrie tabla")
}
fun Sterge_Tabla()
{
println(nume + " sterge tabla")
}
}
class Elev( nume:String) : Om(nume)
{
fun Asculta_De_Profesor()
{
println("Elevul " + nume + " asculta")
}
}
class Profesor( nume:String) : Om(nume)
{
fun Preda_La_elev()
{
println("Profesorul " + nume + " preda")
}
fun Porneste_Calculatoare( calculatoare: MutableList<Calculator>)
{
calculatoare.forEach { it.Start() }
}
}
class Calculator(val id: Int)
{
fun Start()
{
println("S-a pornit calculatorul: " + id)
}
}
class Sala(val nume: String, val elevi: MutableList<Elev>, val p:Profesor, val calculatoare: MutableList<Calculator>)
{
fun Actioneaza(om:String, actiune: String)
{
when(om)
{
"elev"-> {
when(actiune)
{
"asculta"-> elevi.forEach { it.Asculta_De_Profesor() }
"sterge" -> elevi.forEach{it.Sterge_Tabla()}
"scrie" -> elevi.forEach{it.Scrie_Tabla()}
}
}
"profesor"-> {
when(actiune)
{
"preda"->p.Preda_La_elev()
```

```
"sterge" -> p.Sterge_Tabla()
"scrie" -> p.Scrie_Tabla()
"start" -> p.Porneste_Calculatoare(calculatoare)
}
}
}
}
}
fun main(args:Array<String>) {
val alex= Elev("Alex")
val marian= Elev("Marian")
val hara= Elev("Hara")
val m= mutableListOf<Elev>()
m.add(alex)
m.add(hara)
m.add(marian)
val c1=Calculator(1)
val c2=Calculator(2)
val c3=Calculator(3)
val calculatoare = mutableListOf<Calculator>()
calculatoare.add(c1)
calculatoare.add(c2)
calculatoare.add(c3)
val p=Profesor("George")
val sala=Sala("Sala2",m,p,calculatoare)
sala.Actioneaza("elev","asculta")
sala.Actioneaza("profesor","sterge")

}
//sv76 kot106
```

---

persoana_utilizator_agenda_MutableList

```
interface Person
{
fun getName():String
fun getAge():Int
}

class PersonImplementation(val nume:String, val ani:Int):Person
{
override fun getAge(): Int {
return ani
}
override fun getName(): String {
return nume
}
}

class Utilizator(val person:Person):Person by person
{
override fun getAge(): Int {
return person.getAge()
}
override fun getName(): String {
return person.getName()
}
}

class Agenda(val persoane:MutableList<Utilizator>)
{
fun SearchThrough(name:String): String?
{
persoane.forEach { if(it.getName()==name) { return name } }
return null
}
}

fun main(args:Array<String>) {
val alex = PersonImplementation("Alex", 20)
val vlad = PersonImplementation("vlad", 21)
val tudor = PersonImplementation("tudor", 22)
val alin = PersonImplementation("Alin", 23)
var users= mutableListOf<Utilizator>(
Utilizator(alex),
Utilizator(vlad),
Utilizator(tudor),
Utilizator(alin)
)
val agenda = Agenda(users)
println(agenda.SearchThrough("Alex"))
println(agenda.SearchThrough("Ionut"))
}
//////////////////
```

ST23,Kot635/restanta2/Stefania

```
interface Person
{
fun getName():String
fun getAge():Int
}

class Utilizator(val nume:String, val ani:Int):Person
{
override fun getAge(): Int {
return ani
}

override fun getName(): String {
return nume
}
}

class Agenda(val persoane:MutableList<Person>)
{
fun SearchThroug(name:String): String?
{
persoane.forEach { if(it.getName()==name) { return name } }
return null
}
}

fun main(args:Array<String>) {


val alex=Utilizator("Alex",20)
val vlad=Utilizator("vlad",21)
val tudor=Utilizator("tudor",22)
val alin=Utilizator("Alin",23)
var persons = mutableListOf<Person>(alex,vlad,tudor,alin)
val agenda = Agenda(persons)
println(agenda.SearchThroug("Alex"))
println(agenda.SearchThroug("Ionut"))
```

---

sortare_cu_arbore

```
class ArboreBinar<T>(var value:T){
var stanga: ArboreBinar<T>?=null
var dreapta:ArboreBinar<T>?=null
fun <U> map(f:(T) -> U):ArboreBinar<U>{
val arbore=ArboreBinar<U>(f(value))
if(stanga!=null) arbore.stanga=stanga?.map(f)
if(dreapta!=null) arbore.dreapta=dreapta?.map(f)
return arbore
}
fun AfisezParteaSuperioara()="(${stanga?.value},$value,${dreapta?.value})"
fun insert(value: T)
{
if(this==null)
{
this = ArboreBinar(value)
}

}
}
fun treeSort(list:List<Int>)
{
var tree = ArboreBinar<Int>(list[0])
var SubList = list.subList(1,list.size)
for (elem in SubList)
{

}


}

fun main()
{
val list= listOf(10,6,20,3,33,8,11)

}
```

---

submultimi_cuvinte_txt_lambda(colectii)

```
import java.io.File
val procesare={ str:String-> str.substring(str.length/2-1,(str.length/2)+1)}
val stergere={str:String-> str.substring(2)}
fun main()
{
val filename="Text.txt"
val file = File(filename)
var exists=file.exists()
if(!exists)
{
println("Fisierul nu exista.Se va crea unul nou")
file.createNewFile()
file.writeText("Salut Lume!\nPlec din aceasta lume.")
}
var cuvinte:MutableList<String> = file.readText().split(" ",".","!","\n",",").toList() as MutableList<String>
cuvinte.removeIf {
it=="" //eliminam spatii libere
}
println(cuvinte)

var final=cuvinte.filter { it.length>3 }.map{procesare(it)}//pt kot39
println(final)
var final2= cuvinte.filter { it.length>3 }.map{stergere(it)}//pt Kot 40
println(final2)
}
```

```
//sv33kot39/ sv32/kot40
```

---

submultimiAB_sumcu5siMulcu7

```
//object functie{
// fun <A, B> pura(b: B) = {_:A ->b}
//}//functie ca un aplicative
fun <A,B,C> ((A) -> B).map(transform:(B) -> C): (A) -> C = {t -> transform(this(t))}//functii extensie
fun <A,B,C> ((A) -> B).flatMap(fm:(B) -> (A) -> C): (A) -> C = {t -> fm(this(t))(t)}
fun <A,B,C> ((A) -> B).ap(fab:(A)->(B) -> C): (A) -> C = fab.flatMap{f -> map(f)}

fun main()
{
val sumCu5SiMulCu7: (Int) -> Int = {i: Int -> i+5}.ap{{j:Int -> j*7}}
println(sumCu5SiMulCu7(7))
println(sumCu5SiMulCu7(8))
println(sumCu5SiMulCu7(9))
}
//kot31

```