

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]
```

```

84.         pool.map(0, print_lists, list)
85.         pool.start_thread(0)
86.         print(f"Thread 0 is alive: {pool.get_thread_state(0)}")
87.         pool.map(1, reverse_print_lists, list)
88.         pool.start_thread(1)
89.         print(f"Thread 1 is alive: {pool.get_thread_state(1)}")
90.         time.sleep(7)
91.         print(f"\nThread 0 is alive: {pool.get_thread_state(0)}")
92.         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.
96. from queue import Queue
97.
98. class Student:
99.     def __init__(self, name, mediator):
100.         self.name = name
101.         self.mediator = mediator
102.
103.     def send_message(self, message):
104.         self.mediator.send_message(self, message)
105.
106.     def receive_message(self, sender, message):
107.         print(f"{self.name} a primit mesajul de la {sender}: {message}")
108.
109. class Professor:
110.     def __init__(self, name, mediator):
111.         self.name = name
112.         self.mediator = mediator
113.
114.     def send_message(self, message):
115.         self.mediator.send_message(self, message)
116.
117.     def receive_message(self, sender, message):
118.         print(f"{self.name} a primit mesajul de la {sender}: {message}")
119.
120. class Mediator:
121.     def __init__(self):
122.         self.queue = Queue()
123.
124.     def send_message(self, sender, message):
125.         self.queue.put((sender, message))
126.
127.     def start_processing(self):
128.         while not self.queue.empty():
129.             sender, message = self.queue.get()
130.             receiver = None
131.
132.             # Determină destinatarul în funcție de tipul expeditorului
133.             if isinstance(sender, Student):
134.                 receiver = professor
135.             elif isinstance(sender, Professor):
136.                 receiver = student
137.
138.             # Transmitere mesaj către destinatar
139.             if receiver:
140.                 receiver.receive_message(sender.name, message)
141.
142. # Exemplu de utilizare
143. mediator = Mediator()
144.
145. student = Student("John", mediator)
146. professor = Professor("Profesorul", mediator)
147.
148. mediator.student = student
149. mediator.professor = professor
150.
151. student.send_message("Buna ziua!")
152. professor.send_message("Salutare!")
153.
154. mediator.start_processing()
155.

```

problema 4 - Python

156. *###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.###*

```
157.
158. from math import sqrt
159. from functional import seq
160. from threading import Thread
161.
162. min_fun = lambda list: seq(list).min()
163. max_fun = lambda list: seq(list).max()
164. mean_fun = lambda list: seq(list).sum() / seq(list).len()
165. square_mean_fun = lambda list: sqrt(seq(list).map(lambda it: it * it).sum() / seq(list).len())
166. subsequence_fun = lambda list: seq(list).filter(lambda it: -square_mean_fun(list) <= it <= square_mean_fun(list))
167. if __name__ == "__main__":
168.     lists = []
169.     numbers = seq(open("output.txt", "r").readline().split(' ')) \
170.         .map(lambda it: it.replace('\t', ' ')) \
171.         .map(lambda it: int(it))
172.     while numbers.len() > 0:
173.         if numbers.len() >= 10:
174.             lists.append(numbers.take(10).list())
175.             numbers = numbers.drop(10)
176.         else:
177.             lists.append(numbers.take(numbers.len()).list())
178.             numbers = numbers.drop(numbers.len())
179.
180.     results = [] # min,max,mean,subsequence
181.     threads = []
182.     for list in lists:
183.         aux = Thread(target=lambda list, results: results.append(
184.             (min_fun(list), max_fun(list), mean_fun(list), subsequence_fun(list))), args=(list, results,))
185.         aux.start()
186.         threads.append(aux)
187.     for thread in threads:
188.         thread.join()
189.
190.     i=0
191.     for i in range(len(lists)):
192.         print(f"Sequence:{lists[i]}, min:{results[i][0]}, max:{results[i][1]}, mean:{results[i][2]}, subsequence:{results[i][3]}")
193.     i = 0
194.
```

problema 6-Kotlin

Pornind de la exemplul din curs cu functori none si some, sa se creeze in Kptlin o transformare(funcutor) 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.###

```
195.
196. sealed class TipSimplu<out T>{
197.     object None:TipSimplu<Nothing>(){
198.         override fun toString()="Cu None"
199.     }
200.     data class Some<out T>(val value:T):TipSimplu<T>()
201.     companion object
202. }
203.
204.
205. fun <T>TipSimplu<T>.toInt():Int=when(this){
206.     TipSimplu.None->-1
207.     is TipSimplu.Some->(0..1).random()
208. }
209.
210. fun TipSimplu<Int>.map(transform:(TipSimplu<Int>->Int):TipSimplu<Int>{
211.     return TipSimplu.Some<Int>(transform(this))
212. }
213.
214. fun main()
215. {
216.     println(TipSimplu.None.map(TipSimplu<Int>::toInt))
217.     println(TipSimplu.Some(3).map(TipSimplu<Int>::toInt))
218.     println(TipSimplu.Some(5).map(TipSimplu<Int>::toInt))
219. }
220.
221. ---Problema 18
222. ###Sa se creeze in Py un program care primeste un sir de cautare si mai multe nume de fisiere text(inclusiv sursa program).Aplicatia cauta 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.###
223.
224. import re
225.
226. def main():
```

```

227. search_text=input("sirul care trebuie cautat=")
228. number_of_files=int(input("numarul de fisiere="))
229. fnames=[]
230. for _ in range(number_of_files):
231.     file_name=input("nume fisier=")
232.     fnames.append(file_name)
233.
234. replace_flag=input("doresti sa inlocuiesti cu ceva textul(y/n)?\n").lower() == 'y'
235. replace_text=""
236. if replace_flag:
237.     replace_text=input("noul text=")
238.
239. for name in fnames:
240.     file_name_written=False
241.     replaced_lines=[]
242.
243.     with open(name,"r") as file:
244.         lines=file.readlines()
245.         replaced_lines=lines.copy()
246.
247.         for line in lines:
248.             if search_text in line:
249.                 if not file_name_written:
250.                     file_name_written=True
251.                     print(f"inside {name}")
252.                     index=lines.index(line)
253.                     print(f"at line {index}")
254.                     if replace_flag:
255.                         replaced_lines[index]=line.replace(search_text,replace_text)
256.
257.         #this is bs
258.         if replace_flag:
259.             with open(name,"w") as file:
260.                 file.writelines(replaced_lines)
261. main()
262.

```

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.##

```

263. from multiprocessing import Process, Queue
264. from functional import seq
265.
266.
267. def State0(q):
268.     l = q.get()
269.     if len(l) > 0:
270.         q.put(True)
271.         q.put(l)
272.     else:
273.         q.put(False)
274.     time.sleep(1)
275.
276.
277.
278. def State1(q):
279.     l: list = q.get()
280.
281.     if seq(l).filter(lambda it:it%2==0).len()>0:
282.         aux = seq(l).filter(lambda it: it % 2 == 0).first()
283.         l.remove(aux)
284.         print(f"State 1 a eliminat {aux}")
285.         q.put(True)
286.         q.put(l)
287.         # time.sleep(1)
288.
289.
290. def State2(q):
291.     l: list = q.get()
292.     if seq(l).filter(lambda it:it%2!=0).len()>0:
293.         aux = seq(l).filter(lambda it: it % 2 != 0).first()
294.         l.remove(aux)
295.         print(f"State 2 a eliminat {aux}")
296.         q.put(True)
297.         q.put(l)
298.         # time.sleep(1)
299.
300.
301. class FSM:
302.     def __init__(self, list):
303.         self.__queue = Queue()
304.         self.__queue.put(True)
305.         self.__queue.put(list)
306.         self.__states=[State0,State1,State2]

```

```

307.         self.__current_state = 0
308.
309.     def start(self):
310.         while self.__queue.get():
311.             process = Process(target=self.__states[self.__current_state], args=(self.__queue,))
312.             process.start()
313.             self.__current_state = (self.__current_state + 1) % 3
314.             process.join()
315.
316.
317. if __name__ == "__main__":
318.     l = list(range(10))
319.     fsm = FSM(l)
320.     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. ###

```

321.
322. from abc import ABCMeta, abstractmethod
323.
324.
325. class MamiferAbstraction(metaclass=ABCMeta):
326.     def __init__(self, implementation):
327.         self._implementation = implementation
328.
329.     @abstractmethod
330.     def speak(self):
331.         pass
332.
333.     @abstractmethod
334.     def preffered_food(self):
335.         pass
336.
337.
338. class Mamifer(MamiferAbstraction):
339.     def __init__(self, implementation):
340.         super().__init__(implementation)
341.
342.     def speak(self):
343.         self._implementation.speak()
344.
345.     def preffered_food(self):
346.         self._implementation.preffered_food()
347.
348.
349. class MamiferImplementation(metaclass=ABCMeta):
350.     @abstractmethod
351.     def speak(self):
352.         pass
353.
354.     @abstractmethod
355.     def preffered_food(self):
356.         pass
357.
358.
359. class Femeie(MamiferImplementation):
360.     def speak(self):
361.         print("Vorbeste cu voce de femeie.")
362.
363.     def preffered_food(self):
364.         print("Caviar")
365.
366.
367. # 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...
368. class Barbat(MamiferImplementation):
369.     def speak(self):
370.         print("Vorbeste cu voce de barbat.")
371.
372.     def preffered_food(self):
373.         print("Mici")
374.
375.
376. class Caine(MamiferImplementation):
377.     def speak(self):
378.         print("Ham ham.")
379.
380.     def preffered_food(self):
381.         print("Carne")
382.
383.
384. class Pisica(MamiferImplementation):
385.     def speak(self):
386.         print("Miau miau.")
387.
388.     def preffered_food(self):

```

```

389.         print("Peste")
390.
391. if __name__ == '__main__':
392.     caine=Mamifer(Caine())
393.     pisica=Mamifer(Pisica())
394.     femeie=Mamifer(Femeie())
395.     barbat=Mamifer(Barbat())
396.     mamifere=[caine,pisica,femeie,barbat]
397.     for mamifer in mamifere:
398.         mamifer.speak()
399.         mamifer.preffered_food()
400.
401.

```

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

```

402.
403. from more_itertools import flatten, map_reduce
404. from functional import seq
405.
406.
407. def read_words(files):
408.     words = []
409.     for file in files:
410.         aux = open(file).read().split(' ')
411.         aux = [(word, file) for word in aux]
412.         words.append(aux)
413.     words = list(flatten(words))
414.     return words
415.
416.
417. if __name__ == '__main__':
418.     files = ["file1.txt", "file2.txt"]
419.     words = read_words(files)
420.     key_func = lambda pair: pair[0]
421.     value_func = lambda pair: (pair[1], 1)
422.     reduce_func = lambda list_of_pairs: seq(list_of_pairs).reduce_by_key(lambda x, y: x + y)
423.     dict = dict(map_reduce(words, keyfunc=key_func, valuefunc=value_func, reducefunc=reduce_func))
424.     print('{}')
425.     for pair in dict.items():
426.         print(f"\'{pair[0]}\': {pair[1]}")
427.     print('{}')
428.     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

429. from functional import seq
430.
431. class Originator:
432.     def __init__(self, l):
433.         self.__state = l
434.
435.     def set_state(self, l):
436.         self.__state = l
437.
438.     def get_state(self):
439.         return self.__state
440.
441.     def save_state_to_memento(self):
442.         return Memento(self.__state)
443.
444.     def restore_state_from_memento(self, memento):
445.         self.__state = memento.get_state()
446.
447.
448. class Memento:
449.     def __init__(self, state):
450.         self.__state = state
451.
452.     def get_state(self):
453.         return self.__state
454.
455.
456. class Caretaker:
457.     def __init__(self):
458.         self.__states = []
459.
460.     def add(self, state):
461.         self.__states.append(state)
462.
463.     def get(self):
464.         rez = self.__states[-1]
465.         self.__states.remove(self.__states[-1])

```

```

466.         return rez
467.
468.
469.
470. if __name__ == '__main__':
471.     numbers=open("file1.csv",'r').read().split(',')
472.     numbers=seq(numbers).map(lambda it:int(it)).list()
473.
474.     f1=lambda x:(x%2==0 and x+1) or x
475.     f2=lambda x:3*x*x-2*x+1
476.     f3=lambda x:x+x+1
477.     functions=[f1,f2,f3]
478.
479.     caretaker=Caretaker()
480.     originator=Originator(numbers)
481.
482.     print(f"Lista originala: {numbers}")
483.     for i,f in enumerate(functions,1):
484.         caretaker.add(originator.save_state_to_memento())
485.         originator.set_state(seq(originator.get_state()).map(f).list())
486.         print(f"Lista dupa aplicarea functiei f{i}: {originator.get_state()}")
487.         restore=input("Doriti restaurarea starii anterioare a listei? 1-Da Altceva-Nu\nRaspunsul dumneavoastra: ")
488.         if restore=="Da":
489.             originator.restore_state_from_memento(caretaker.get())
490.         print(f"Lista finala: {originator.get_state()}")
491.

```

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.###

```

492. from functional import seq
493. import asyncio
494. import re
495.
496. coroutines_per_processing = 4
497.
498.
499. async def _elimin_spatii_multiple(partition: str):
500.     return re.sub("[ ]{2,}", ' ', partition)
501.
502.
503. async def elimin_spatii_multiple(text):
504.     # text = seq(open(file, 'r').readlines())
505.     partition_len = text.len() // coroutines_per_processing + 1
506.     partitions = []
507.     while text.len() > 0:
508.         partitions.append(text.take(partition_len).make_string(''))
509.         text = text.drop(partition_len)
510.     tasks = [asyncio.create_task(_elimin_spatii_multiple(partition)) for partition in partitions]
511.     new_text = ''.join([await task for task in tasks])
512.     return new_text
513.
514.
515. async def _elimin_salturi_linie_noua(partition: str):
516.     return partition.replace('\n', ' ')
517.
518.
519. async def elimin_salturi_linie_noua(text):
520.     partition_len = text.len() // coroutines_per_processing + 1
521.     partitions = []
522.     while text.len() > 0:
523.         partitions.append(text.take(partition_len).make_string(''))
524.         text = text.drop(partition_len)
525.     tasks = [asyncio.create_task(_elimin_salturi_linie_noua(partition)) for partition in partitions]
526.     new_text = ''.join([await task for task in tasks])
527.     return new_text
528.
529.
530. async def main():
531.     file = "economics_to_be_happier.txt"
532.     text = seq(open(file, 'r',encoding='utf-8').readlines())
533.     task = asyncio.create_task(elimin_spatii_multiple(text))
534.     new_text = await task
535.     print(f"Dupa eliminarea spatiilor multiple:\n{new_text}")
536.     task = asyncio.create_task(elimin_salturi_linie_noua(seq(new_text)))
537.     new_text = await task
538.     print(f"Dupa eliminarea salturilor la linie noua:\n{new_text}")
539.     with open("OUTPUT.txt", 'w') as output:
540.         output.write(new_text)
541.
542.
543. asyncio.run(main())

```

```

1.  asyncio
2.  -----
3.  import asyncio
4.  #in loc de @asy.coroutine folosim async pt a declara o corutina
5.  async def doSum(inRange):#folosim async pt a avea acces la functiile asyncio
6.  print( "Incepem sa adunam {} numere".format(inRange) )
7.  sum = 0
8.  chart = 10000
9.  for i in range(inRange):
10. sum = sum + i
11. # print( i);
12. if i > chart:
13. chart = chart + 100000
14. await asyncio.sleep(0)#asteptam (timp scris intre paranteze) unitati de timp cand i> chart
15. print( "Terminam de adunat {} numere in rezultatul {}".format(inRange, sum) )
16. return sum
17.
18. async def main(queue):#tot o corutina
19. while len(queue) != 0:
20. if len(asyncio.all_tasks()) < 5:#un set de task-uri neterminate inca
21. a = loop.create_task( doSum(queue.pop()) )#creem un task nou axat pe functia doSum
22. await asyncio.sleep(0)
23. await asyncio.wait([a])#se ruleaza obiecte asteptate in iterabilul [a] simultan si se blocheaza pana la conditia specificata de return_when.
24. # print(a.result())
25.
26. if __name__ == '__main__':
27. queue = []
28. queue.append(20000000)
29. queue.append(100000)
30. queue.append(30044440)
31. queue.append(266660000)
32. queue.append(1066600)
33. queue.append(3333300)
34. queue.append(20888000)
35. queue.append(1444000)
36. queue.append(3004440)
37. queue.append(20888000)
38. queue.append(14666600)
39.
40. loop = asyncio.get_event_loop()#returneaza loop-ul curent de lucru
41. loop.run_until_complete( main(queue) ) #loop-ul ruleaza pana cand se termina corutina main
42. # loop.close()
43. #"""Sa se realizeze calculul simultan pentru patru valori diferite ale lui n luate dintr-o
44. #coada de catre patru corutine diferite (se va utiliza modulul asyncio)."""
45.
46.
47.
48. -----

```

automat_3stari_expresiilambda

```

49. -----
50. from functional import seq
51.
52. class StateZero:
53. def __init__(self,automat):
54. self.automat=automat
55. def operation(self):
56. if(seq(self.automat.lista).non_empty()):
57. print(self.automat.lista)
58. self.automat.current_state=self.automat.states[1]
59. else:
60. self.automat.keep_running=False
61. print("lista vida")
62.
63. class StateOne:
64. def __init__(self,automat):
65. self.automat=automat
66. def operation(self):
67. if(seq(self.automat.lista).exists(lambda number:number%2==0)):
68. self.automat.lista.remove(seq(self.automat.lista).\
69. find(lambda number: number%2==0))
70. self.automat.current_state=self.automat.states[2]
71. class StateTwo:
72. def __init__(self,automat):
73. self.automat=automat
74. def operation(self):
75. if(seq(self.automat.lista).exists(lambda number: number%2==1)):
76. 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_decorator
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

```

```

163. def component(self):
164.     return self._camera
165. def operation(self):
166.     return self._camera.operation()
167.
168. class ConcreteDecoratorA(Decorator):
169.     def operation(self):
170.     print(f'In bar au fost urmatoarele cheltuieli: {self.alcohol_price}')
171.     # return "DecoratedCamera"
172.
173. def genereare_plata(component: Camera):
174.     component.operation()
175.
176. if __name__ == '__main__':
177.     '''
178.     Componenta
179.     ComponentaConcreta == Camera
180.     Decorator
181.     DecoratorConcretA = Decorator specific, pot face A, B, C... dar sa aiba implementari diferite == Consum_Bar
182.     '''
183.
184.     # camere concrete
185.     camera1 = ConcreteCamera()
186.     camera2 = ConcreteCamera()
187.     camera3 = ConcreteCamera()
188.
189.     # inainte de a decora camerele cu bar
190.     camere_hotel = [camera1, camera2, camera3]
191.     for camera in camere_hotel:
192.         genereare_plata(camera)
193.     print()
194.
195.     # decorez camera2 cu bar
196.     camera_decorata1 = ConcreteDecoratorA(camera1, 24)
197.     camera_decorata2 = ConcreteDecoratorA(camera2, 0)
198.     camere_hotel[0] = camera_decorata1
199.     camere_hotel[1] = camera_decorata2
200.
201.     for camera in camere_hotel:
202.         genereare_plata(camera)
203.     -----
204.     comanda_student,colega
205.     -----
206.     from enum import Enum
207.
208.     class Stari(Enum):
209.         FERICIT = 1
210.         DISPERAT = 2
211.         MORT = 3
212.         STIE_POPA_CA_UMBILI_DEZGROPAT = 4
213.
214.     class Command:
215.     def execute(self, student):
216.         pass
217.
218.     class Dispera(Command):
219.     def execute(self, student):
220.         student.stareSufleteasca(Stari.DISPERAT)
221.
222.     class Rupe(Command):
223.     def execute(self, student):
224.         student.stareSufleteasca(Stari.MORT)
225.
226.     class Maxima(Command):
227.     def execute(self, student):
228.         student.stareSufleteasca(Stari.STIE_POPA_CA_UMBILI_DEZGROPAT)
229.
230.     class Bucurie(Command):
231.     def execute(self, student):
232.         student.stareSufleteasca(Stari.FERICIT)
233.
234.     class Student:
235.     def __init__(self):
236.         self.stare = Stari.FERICIT
237.         def stareSufleteasca(self, stare):
238.             self.stare = stare
239.         def printStare(self):
240.             print("Saracul student se simte: " + self.stare.name)
241.
242.     class Colega:
243.     def __init__(self, baiat):
244.         self.student = baiat
245.         self.comanda = Bucurie()
246.         def setComanda(self, comanda):
247.             self.comanda = comanda
248.         def executa(self):
249.             self.comanda.execute(self.student)
250.
251.     def main():
252.         un_student = Student()
253.         o_colega = Colega(un_student)
254.         un_student.printStare()

```

```

255.
256. o_colega.setComanda(Dispera())
257. o_colega.executa()
258. un_student.printStare()
259.
260. o_colega.setComanda(Rupe())
261. o_colega.executa()
262. un_student.printStare()
263.
264. o_colega.setComanda(Maxima())
265. o_colega.executa()
266. un_student.printStare()
267.
268. if __name__ == '__main__':
269.     main()
270.
271. -----

```

comanda(model)_student, profesor

```

272. -----
273. from enum import Enum
274.
275.
276. class Stari(Enum):
277.     FERICIT = 1
278.     DISPERAT = 2
279.     MORT = 3
280.     STIE_POPA_CA_UMBLI_DEZGROPAT = 4
281.
282.
283. class Command:
284.     def execute(self, student):
285.         pass
286.
287.
288. class Dispera(Command):
289.     def execute(self, student):
290.         student.stareSufleteasca(Stari.DISPERAT)
291.
292.
293. class Rupe(Command):
294.     def execute(self, student):
295.         student.stareSufleteasca(Stari.MORT)
296.
297.
298. class Maxima(Command):
299.     def execute(self, student):
300.         student.stareSufleteasca(Stari.STIE_POPA_CA_UMBLI_DEZGROPAT)
301.
302.
303. class Bucurie(Command):
304.     def execute(self, student):
305.         student.stareSufleteasca(Stari.FERICIT)
306.
307.
308. class Student:
309.     def __init__(self):
310.         self.stare = Stari.FERICIT
311.
312.     def stareSufleteasca(self, stare):
313.         self.stare = stare
314.
315.     def printStare(self):
316.         print("Saracul student se simte: " + self.stare.name)
317.
318.
319. class Profesor:
320.     def __init__(self, baiat):
321.         self.student = baiat
322.         self.comanda = Dispera()
323.
324.     def setComanda(self, comanda):
325.         self.comanda = comanda
326.
327.     def executa(self):
328.         self.comanda.execute(self.student)
329.
330.
331. if __name__ == '__main__':
332.     un_student = Student()
333.     prof = Profesor(un_student)
334.     un_student.printStare()
335.
336.     prof.setComanda(Rupe())
337.     prof.executa()
338.     un_student.printStare()
339.
340.     prof.setComanda(Maxima())

```

```

341. prof.executa()
342. un_student.printStare()
343.
344. prof.setComanda(Bucurie())
345. prof.executa()
346. un_student.printStare()

```

fabrica_de_fabrici_limba_buton_tkinter

```

347. import abc
348. from tkinter import *
349.
350. class Displayer(Frame):
351.     def __init__(self, root):
352.
353.     self.root = root
354.     self.canvas = Canvas(root)
355.
356. class Factory_Default(metaclass=abc.ABCMeta):
357.     @abc.abstractmethod
358.     def UIdisplay(self):
359.     pass
360.
361. class Factory1_English(Factory_Default):
362.     def __init__(self, name):
363.     self.name = name
364.     def UIdisplay(self):
365.     print(self.name)
366.     self.root = Tk()
367.     self.root.title("english factory")
368.     self.disp = Displayer(self.root)
369.     self.disp.button = Button(self.root, text="Button one", padx=20, pady=20).grid(row=0, column=1)
370.     self.disp.button = Button(self.root, text="Button two", padx=20, pady=20).grid(row=0, column=2)
371.     self.disp.button = Button(self.root, text="Button three", padx=20, pady=20).grid(row=0, column=3)
372.     self.root.mainloop()
373.
374. class Factory2_Romanian(Factory_Default):
375.     def __init__(self, name):
376.     self.name = name
377.     def UIdisplay(self):
378.     print(self.name)
379.     self.root = Tk()
380.     self.root.title("fabrica romaneasca")
381.     self.disp = Displayer(self.root)
382.     self.disp.button = Button(self.root, text="Buton unu", padx=20, pady=20).grid(row=0, column=1)
383.     self.disp.button = Button(self.root, text="Buton doi", padx=20, pady=20).grid(row=0, column=2)
384.     self.disp.button = Button(self.root, text="Buton trei", padx=20, pady=20).grid(row=0, column=3)
385.     self.root.mainloop()
386.
387. class LanguageFactory:
388.     def createFactory(self, name):
389.     if name == 'english':
390.     return Factory1_English(name)
391.     elif name == 'romanian':
392.     return Factory2_Romanian(name)
393.
394. if __name__ == '__main__':
395.     FabricaMare = LanguageFactory()
396.     type = input("introduce ui language romanian/english")
397.     FabricaMica = FabricaMare.createFactory(type)
398.     FabricaMica.UIdisplay()

```

fatada_boombox_SOLID

```

class play:
399.     '''Subsystem # 1'''
400.     def play(self):
401.     print("playing...")
402.
403.     class record:
404.     '''Subsystem # 2'''
405.     def record(self):
406.     print("recording...")
407.
408.     class fastforward:
409.     '''Subsystem # 3'''
410.     def fastforward(self):
411.     print("fast forwarding...")
412.
413.     class rewind:
414.     '''Subsystem # 4'''

```

```

415. def rewind(self):
416.     print("rewinding...")
417.
418. class radio:
419.     '''Subsystem # 5'''
420.     def radio(self):
421.         print("switching to radio...")
422.
423. class volumeup:
424.     '''Subsystem # 6'''
425.     def volumeup(self):
426.         print("volume level up...")
427.
428. class volumedown:
429.     '''Subsystem # 7'''
430.     def volumedown(self):
431.         print("volume level down...")
432.
433. class batterystats:
434.     '''Subsystem # 8'''
435.     def batterystats(self):
436.         print("100% of your life...")
437.
438. class Boombox:
439.     '''Facade'''
440.     def __init__(self):
441.         #initializare metode n shit
442.         self.playing = play()
443.         self.recording = record()
444.         self.fastforwarding = fastforward()
445.         self.rewind = rewind()
446.         self.radio = radio()
447.         self.volumeup = volumeup()
448.         self.volumedown = volumedown()
449.         self.batterystatus = batterystats()
450.     def startplaying(self):
451.         self.volumeup.volumeup()
452.         self.playing.play()
453.     def startrecording(self):
454.         self.volumeup.volumeup()
455.         self.recording.record()
456.         self.playing.play()
457.     def switchtoradio(self):
458.         self.radio.radio()
459.         self.playing.play()
460.     def volumetoggle(self):
461.         self.volumeup.volumeup()
462.         self.volumedown.volumedown()
463.     def batterystats(self):
464.         self.batterystatus.batterystats()
465.
466. if __name__ == "__main__":
467.     music = Boombox()
468.     music.startplaying()
469.
470.     music.startrecording()
471.     music.batterystats()
472.
473.     music.volumetoggle()
474.
475.     music.switchtoradio()

```

----- flatten_inregistrare -----

```

476. inregistrare={'Nume':'Bula','Locatia':{'Oras':'Pocreaca','Tara':'ROM'},'hobi':['Manea','Bautura','Femei']}
477. def flatten_tabel(y):
478.     iesire={}
479.     def flatten(x,nume=''):
480.         if type(x) is dict:
481.             for a in x:
482.                 flatten(x[a],nume+a+'_')#recursiv
483.         elif type(x) is list:
484.             i=0
485.             for a in x:
486.                 flatten(a,nume+str(i)+'_')
487.                 i+=1
488.         else:
489.             iesire[nume[:-1]]=x#afisare totala la nume
490.             flatten(y)
491.     return iesire
492. listaCurata=flatten_tabel(inregistrare)
493. print(listaCurata)
494. -----
495. grafic_calcul_tkinter
496. -----
497. import tkinter as tk

```

```

498. if __name__ == '__main__':
499.     frame = tk.Tk()
500.     frame.geometry("500x1500")
501.     canvas=tk.Canvas(frame,width=1000,height=1000)
502.     canvas.place(x=0,y=0)
503.
504.     n0=0
505.     x=[]
506.     y=[]
507.     for i in range(21,50):
508.         x.append(i)
509.         s=n0
510.         for j in range(0,i):
511.             s=s+j
512.         y.append(s)
513.
514.         print(x)
515.         print(y)
516.
517.     T= tk.Text(frame, height=5,width=100)
518.     T.insert(tk.END,"i=")
519.     T.insert(tk.END, x)
520.     T.insert(tk.END, "\nS=")
521.     T.insert(tk.END, y)
522.     T.pack()
523.
524.     for i in range(0,len(x)-2):
525.         if(i%2):
526.             canvas.create_line(x[i], y[i], x[i+1], y[i+1],fill='red',width=4)
527.         else:
528.             canvas.create_line(x[i], y[i], x[i + 1], y[i + 1],fill='blue',width=4)
529.     frame.mainloop()

```

intertool_closure_fisiere_temporare

```

530. import itertools as it
531. #closure ca un contor
532. def fabricaDeContoare(prefix):
533.     iterator=it.count(start=0,step=1)
534.     contorvar=next(iterator) #0
535.     def contor():
536.         nonlocal iterator,contorvar
537.         contorvar=next(iterator)
538.         nume1=prefix+str(contorvar)
539.         nume2 =prefix + str(contorvar+1)
540.         creeazaFisier(nume1,nume2)
541.         return contorvar
542.     def valoareaCurentaContor():
543.         nonlocal iterator,contorvar
544.         return contorvar
545.     def creeazaFisier(nume1,nume2):
546.         numecomplet=nume1+"-index-"+nume2+".tmp"
547.         f=open(numecomplet,"w+")
548.         f.write("%d" %(valoareaCurentaContor()))
549.         f.close()
550.     def numeUrmator():
551.         nume1 = prefix + str(contorvar+1)
552.         nume2 = prefix + str(contorvar + 2)
553.         return nume1 + "-index-" + nume2 + ".tmp"
554.     return contor,valoareaCurentaContor,numeUrmator
555.     contor,valoareaContor,nextName=fabricaDeContoare("s")
556.     print(contor())
557.     print(contor())
558.     print(contor())
559.     print(valoareaContor())
560.     print(nextName())
561.     print(contor())
562. #SV32Pyt356

```

lant_responsabilitati_nivelalerta_politie_nato

```

563. import abc
564. class Handler(metaclass=abc.ABCMeta):
565.     def __init__(self,successor=None):
566.         self.successor=successor
567.     def handle(self,request):
568.         res=self.check_alertlevel(request)
569.         if not res and self.successor:

```

```

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.             "I 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

```

```

651. def __init__(self):
652.     print("Am creat o femeie.")
653. def ceva(self):
654.     print("Femeia a facut ceva.")
655.
656. class Om(Mamifer):#ConcreteComponent
657.     def __init__(self):
658.         print("Am creat un om.")
659.     #Decorator
660.     class Locare(Mamifer):
661.         def __init__(self,mamifer):
662.             self.mamifer = mamifer
663.         def ceva(self):
664.             self.mamifer.ceva()
665.         print("Am pus mamiferul la locul lui.")
666.
667.     class MamiferHolder:
668.         def __init__(self,mamifer):
669.             self.local_mamifer = mamifer
670.             self.local_mamifer.ceva()
671.         self.decorated = Locare( self.local_mamifer)
672.         self.decorated.ceva()
673.
674. if __name__ == '__main__':
675.     a = MamiferHolder(Femeie())

```

mamifer2_decorator

```

676. from enum import Enum
677. class Decorator:
678.     def addDecorabil(self, mamifer):
679.         self.decorat = mamifer
680.     def interactiune(self):
681.         pass
682.
683. class Mamifer:
684.     pass
685.
686. class AnimalDeCompanie(Mamifer):
687.     pass
688.
689. class Caine(AnimalDeCompanie):
690.     def __init__(self):
691.         print("Ham!")
692.
693. class Pisica(AnimalDeCompanie):
694.     def __init__(self):
695.         print("Meow!")
696.
697. class Femeie(Mamifer, Decorator):#concreteDecorator
698.     def __init__(self):
699.         self.animal = AnimalDeCompanie()
700.         print("Am creat o femeie.")
701.
702.     def amAnimal(self, animal):
703.         self.animal = animal
704.
705.     def interactiune(self):
706.         if isinstance(self.decorat, Om):#daca decorat este instanta a clasei Om
707.             print("Femeie interactiune cu Om")
708.
709.     class Om(Mamifer):
710.         def __init__(self):
711.             self.animal = AnimalDeCompanie()
712.             print("Am creat un om.")
713.         def amAnimal(self, animal):
714.             self.animal = animal
715.
716. if __name__ == '__main__':
717.     femeie = Femeie()
718.     femeie.amAnimal(Caine())
719.     femeie.addDecorabil(Om())
720.     femeie.interactiune()

```

mediator_furnica_SOLID

```

721. import java.io.File
722. val procesare={ str:String-> str.substring(str.length/2-1,(str.length/2)+1)}
723. val stergere={str:String-> str.substring(2)}
724. fun main()
725. {
726.     val filename="Text.txt"

```



```

727. val file = File(filename)
728. var exists=file.exists()
729. if(!exists)
730. {
731.     println("Fisierul nu exista.Se va crea unul nou")
732.     file.createNewFile()
733.     file.writeText("Salut Lume!\nPlec din aceasta lume.")
734. }
735. var cuvinte:MutableList<String> = file.readText().split(" ",".", "!","\\n",",").toList() as MutableList<String>
736. cuvinte.removeIf {
737.     it==" " //eliminam spatii libere
738. }
739. println(cuvinte)
740.
741. var final=cuvinte.filter { it.length>3 }.map{procesare(it)}//pt kot39
742. println(final)
743. var final2= cuvinte.filter { it.length>3 }.map{stergere(it)}//pt Kot 40
744. println(final2)
745. }
746. #sv33py341

```

----- meidator_student_profesor -----

```

747. from abc import ABC
748. import multiprocessing as mp
749.
750. message_queue = mp.Queue()
751.
752. class Mediator(ABC):
753.     def notify(self, sender, event):
754.         pass
755.
756. class ConcreteMediator(Mediator):
757.     def __init__(self, student, profesor):
758.         self.student = student
759.         self.student.mediator = self#se apeleaza mediator.setter
760.         self.profesor = profesor
761.         self.profesor.mediator = self
762.     def notify(self, sender, event):
763.         if event == "Give messages to students":
764.             if message_queue.empty() is True:
765.                 print("Studentii nu au primit mesaje.\n")
766.             else:
767.                 print("Mesajele primite de la profesor sunt:")
768.                 while message_queue.empty() is False:
769.                     print(message_queue.get())
770.                 print()
771.             elif event == "Give messages to teacher":
772.                 if message_queue.empty() is True:
773.                     print("Profesorul nu a primit mesaje.\n")
774.                 else:
775.                     print("Mesajele primite de la studentii sunt:")
776.                     while message_queue.empty() is False:
777.                         print(message_queue.get())
778.                     print()
779.
780. class BaseComponent:
781.     def __init__(self, mediator: Mediator = None):
782.         self._mediator = mediator
783.     @property
784.     def mediator(self):
785.         return self._mediator
786.     @mediator.setter
787.     def mediator(self, mediator):
788.         self._mediator = mediator
789.
790. class Student(BaseComponent):
791.     def get_message(self):
792.         self.mediator.notify(self, "Give messages to students")
793.     def say_message(self, message):
794.         message_queue.put(message)
795.
796. class Profesor(BaseComponent):
797.     def get_message(self):
798.         self.mediator.notify(self, "Give messages to teacher") #se apeleaza @property
799.     def say_message(self, message):
800.         message_queue.put(message)
801.
802. if __name__ == "__main__":
803.     student1 = Student()
804.     student2 = Student()
805.     student3 = Student()
806.     profesor = Profesor()
807.     mediator = ConcreteMediator(student1, profesor)
808.     mediator = ConcreteMediator(student2, profesor)

```

```

809. mediator = ConcreteMediator(student3, profesor)
810.
811. profesor.get_message()
812. student2.get_message()
813.
814. student1.say_message("Eu sunt Feri")
815. student1.say_message("Glumeam. Io-te feedback")
816. student2.say_message("Why are you running")
817. student3.say_message("Imi place pisicile")
818.
819. profesor.get_message()
820. profesor.say_message("Multumesc pentru feedback")
821. profesor.say_message("Da")
822.
823. student1.get_message()

```

----- multiprocessing_ciclica_cuvant_fisier_cozi -----

```

824. import multiprocessing
825.
826. def worker(q):
827.     print("Sunt procesul %s si am primit %s " %(multiprocessing.current_process().name,q.get()))
828.
829. if __name__=='__main__':
830.     f=open("in.txt","r")
831.     a=f.read().split()
832.
833.     queue = multiprocessing.Queue()
834.
835.     p1 = multiprocessing.Process(target=worker, args=(queue,))
836.     p1.start()
837.
838.     p2 = multiprocessing.Process(target=worker, args=(queue,))
839.     p2.start()
840.
841.     p3 = multiprocessing.Process(target=worker, args=(queue,))
842.     p3.start()
843.
844.     for i in a:
845.         queue.put(i)
846.     queue.close()
847.     queue.join_thread()
848.     p1.join()
849.     p2.join()
850.     p3.join()
851. #sv65 py378

```

----- multiprocessing_starmap_iterator -----

```

852. import multiprocessing
853.
854.
855. def count(iterator, k=1):
856.     def f(x, k):
857.         return x + k
858.
859.     contor = 0
860.     for _ in iterator:
861.         contor = f(contor, k)
862.     return contor
863.
864.
865. if __name__ == '__main__':
866.     nume_fisier = 'test.txt'
867.     fisier_citit = []
868.     with open(nume_fisier) as f:
869.         linie = f.readline()
870.         while linie:
871.             fisier_citit.append(linie)
872.             linie = f.readline()
873.
874.     k = 5 # numar procese
875.     numar_linii_per_bucata = len(fisier_citit) // k + 1 # fiecare proces primeste (nr linii / numar procese) linii
876.     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
877.     bucati = [[] for _ in range(k)]
878.     i_bucata = -1 # la linia 30/31 valoarea se va schimba la 0, fiind prima linie executata din for
879.     for i in range(len(fisier_citit)):
880.         if i < numar_linii_per_bucata * (k-1):

```

```

881. if i % numar_linii_per_bucata == 0:
882.     i_bucata += 1
883.     bucati[i_bucata].extend(fisier_citit[i].split()) # fiecare bucata reprezinta o lista de cuvinte
884. else:
885.     bucati[k - 1].extend(fisier_citit[i].split()) # ultima bucata
886.
887. bucati_starmap = list(map(lambda bucata: (bucata,), bucati)) # iteram prin fiecare bucata din bucati si o inlocuim cu tupla formata
    doar din acea bucata
888. # e necesar pentru starmap
889.
890. with multiprocessing.Pool(processes=k) as pool:
891.     results = pool.starmap(count, bucati_starmap)
892.     with open('result.txt', 'w') as f:
893.         f.write(str(sum(results)))
894. #examenAdrianCristea

```

pod_bridge_desena_triunghi_cerc

```

895. import tkinter as tk
896.
897. class DrawApi:#Implementor
898.     def desenare(self,canvas):
899.         pass
900.     class Dreptunghi(DrawApi):#ConcreteImplementor
901.         def desenare(self,canvas):
902.             canvas.create_rectangle(400,200,100,100,fill="blue")
903.
904.     class Cerc(DrawApi):#ConcreteImplementor
905.         def desenare(self,canvas):
906.             canvas.create_oval(90,90,300,300,fill="yellow")
907.
908.     class Triunghi(DrawApi):#ConcreteImplementor
909.         def desenare(self,canvas):
910.             canvas.create_polygon((0, 100, 50, 0, 100, 100), fill="red")
911.
912.     class Shape():#Abstraction
913.         def __init__(self,drawApi):
914.             self.drawApi=drawApi
915.         def Draw(self,canvas):
916.             self.drawApi.desenare(canvas)
917.
918.     if __name__=='__main__':
919.         frame = tk.Tk()
920.         frame.geometry("600x600")
921.         canvas = tk.Canvas(frame, width=600, height=600)
922.         canvas.place(x=0, y=0)
923.
924.         dreptunghi=Shape(Dreptunghi())
925.         cerc=Shape(Cerc())
926.         triunghi=Shape(Triunghi())
927.
928.         dreptunghi.Draw(canvas)
929.         cerc.Draw(canvas)
930.         triunghi.Draw(canvas)
931.
932.         frame.mainloop()
933. #sv76 py314

```

coada_circulara_py_student,colega_kot

```

934. Subiect 1 - Python:
935. Să se implementeze în Python o coadă circulară de procese care scriu
936. într-un fișier. Procesele pot fi puse în așteptare sau pornite (se vor
937. utiliza decoratori). Pentru scrierea în fișier se va asigura coerența
938. utilizând semafoare (se poate crea tot un decorator sau nu).
939.
940.
941. *lipseste partea de wait, al doilea decorator
942. #####
943. import random
944. import multiprocessing as mp
945. from time import sleep
946.
947. semafor = mp.Semaphore(1)
948.
949.

```

```

950. def scrie():
951.     with open("test.txt", "a") as file:
952.         x: str = "Scriu un numar random: "
953.         x += str(random.randint(1, 100))
954.         x += '\n'
955.         with semafor:
956.             print("Am scris")
957.             file.write(x)
958.             sleep(1)
959.             print("Am dormit")
960.             print(x)
961.
962.
963. class CircularQueue(mp.Process):
964.
965.     def __init__(self, size):
966.         self.size = size
967.         self.queue = [None for i in range(size)]
968.         self.front = self.rear = -1
969.         mp.Process.__init__(self)
970.
971.     def enqueue(self, data):
972.
973.         if ((self.rear + 1) % self.size == self.front):
974.             print(" Coadă plină\n")
975.             elif (self.front == -1):
976.                 self.front = 0
977.                 self.rear = 0
978.                 self.queue[self.rear] = data
979.             else:
980.                 self.rear = (self.rear + 1) % self.size
981.                 self.queue[self.rear] = data
982.
983.     def dequeue(self):
984.         if (self.front == -1):
985.             print("Coadă goală\n")
986.             elif (self.front == self.rear):
987.                 temp = self.queue[self.front]
988.                 self.front = -1
989.                 self.rear = -1
990.                 return temp
991.             else:
992.                 temp = self.queue[self.front]
993.                 self.front = (self.front + 1) % self.size
994.                 return temp
995.
996.     def display2(self):
997.         for x in self.queue:
998.             if x is not None:
999.                 pornire(x)
1000.         print()
1001.
1002.
1003.     def porneste(func):
1004.         def wrapper(*args, **kwargs):
1005.             output = func(*args, **kwargs)
1006.             output.start()
1007.             return output
1008.
1009.         return wrapper
1010.
1011.
1012.     @porneste
1013.     def pornire(p: mp.Process):
1014.         print("Am pornit ", p.__str__(), '\n')
1015.         return p
1016.
1017.
1018. if __name__ == '__main__':
1019.     print("hello")
1020.     ob = CircularQueue(5)
1021.     p1 = mp.Process(target=scrie)
1022.     p2 = mp.Process(target=scrie)
1023.     ob.enqueue(p1)
1024.     ob.enqueue(p2)
1025.     ob.display2()
1026.
1027. #####
1028.
1029.

```

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)
```

```

1085. print(student1)
1086. colega.schimba(student1)
1087. print(student1)
1088. }
1089.
1090.
1091. #####3

```

``` ----- proxy_pret_login ----- ```

```

1092. import datetime
1093. class Hacker(Exception):
1094. pass
1095.
1096. class Observator:#Observer
1097. def primesteRata(self, rata):
1098. raise NotImplemented
1099.
1100. class Pret(Observator):#ConcreteObserver
1101. def __init__(self, initial):
1102. self.initial = initial
1103. def primesteRata(self, rata):
1104. self.initial = self.initial - ((rata * self.initial) / 100)
1105. return self.initial
1106.
1107. class Dispatcher:
1108. def __init__(self):
1109. self.observatori = []
1110. self.logger = Logger("Log.txt")
1111. def addObs(self, obs):
1112. self.observatori.append(obs)
1113. def removeObs(self, obs):
1114. self.observatori.remove(obs)
1115. def trimiteNouaRata(self, rata, user):
1116. for obs in self.observatori:
1117. pretNou = obs.primesteRata(rata)
1118. self.logger.log(user.user, rata, pretNou)
1119.
1120. class Cont:
1121. def __init__(self, user, parola):
1122. self.user = user
1123. self.parola = parola
1124.
1125. class Service:#clasa Subject
1126. def operatie(self):
1127. raise NotImplemented
1128.
1129. class IntroducereRata(Service):
1130. def operatie(self):
1131. return int(input('Rata noua: '))
1132.
1133. class Proxy(Service):
1134. def __init__(self):
1135. self.service = None
1136. self.cont = None
1137. self.dispatcher = Dispatcher()
1138. def operatie(self):
1139. return self.service.operatie()
1140. def checkAcces(self):
1141. self.service = Logare()
1142. if self.cont is None:
1143. self.cont = self.operatie()
1144. self.dispatcher.addObs(Pret(int(input('Pret initial: '))))
1145. else:
1146. nouBaiat: Cont = self.operatie()
1147. if not (nouBaiat.user == self.cont.user and nouBaiat.parola == self.cont.parola):
1148. print("Hacker-ule...iesi afara!")
1149. raise Hacker
1150. else:
1151. self.service = IntroducereRata()
1152. rata = self.operatie()
1153. self.dispatcher.trimiteNouaRata(rata, self.cont)
1154.
1155. class Logare(Service):#realSubject
1156. def operatie(self):
1157. return Cont(str(input('User: ')), str(input('Parola: ')))
1158.
1159. class Logger:
1160. def __init__(self, fila):
1161. self.fisier = open(fila, 'a')
1162. def close(self):
1163. self.fisier.close()
1164. def log(self, user, rata, pret):

```

```

1165. self.fisier.write(
1166. 'User-ul ' + user + ' la ' + str(
1167. datetime.datetime.now()) + ' a modificat pretul cu rata de ' + str(rata) + '%, noul pret devenind: ' + str(pret) + '\n')
1168.
1169. if __name__ == '__main__':
1170. proxy = Proxy()
1171. while True:
1172. proxy.checkAcces()
1173. #sv50py335

```

search_sir_replace

```

1174. import re
1175.
1176. def main():
1177. search_text=input("sirul care trebuie cautat=")
1178. number_of_files=int(input("numarul de fisiere="))
1179. fnames=[]
1180. for _ in range(number_of_files):
1181. file_name=input("nume fisier=")
1182. fnames.append(file_name)
1183.
1184. replace_flag=input("doresti sa inlocuiesti cu ceva textul(y/n)?\n").lower() == 'y'
1185. replace_text=""
1186. if replace_flag:
1187. replace_text=input("noul text=")
1188.
1189. for name in fnames:
1190. file_name_written=False
1191. replaced_lines=[]
1192.
1193. with open(name,"r") as file:
1194. lines=file.readlines()
1195. replaced_lines=lines.copy()
1196.
1197. for line in lines:
1198. if search_text in line:
1199. if not file_name_written:
1200. file_name_written=True
1201. print(f"inside {name}")
1202. index=lines.index(line)
1203. print(f"at line {index}")
1204. if replace_flag:
1205. replaced_lines[index]=line.replace(search_text,replace_text)
1206.
1207. #this is bs
1208. if replace_flag:
1209. with open(name,"w") as file:
1210. file.writelines(replaced_lines)
1211. main()

```

strategy_erori_solid

```

1212. import sys
1213.
1214. class Warning(Exception):
1215. pass
1216.
1217. class Error(Exception):
1218. pass
1219.
1220. class GraveError(Exception):
1221. pass
1222.
1223.
1224. class Strategie:
1225. file = None
1226. fileGrav = None
1227. fileWarning = None
1228. @classmethod
1229. def fisiereDeschidere(cls):
1230. fisier_de_log = "Errori.txt"
1231. cls.file = open(fisier_de_log, "w")
1232. fisier_de_log_rau = "ErroriGrave.txt"
1233. cls.fileGrav = open(fisier_de_log_rau, "w")
1234. fisier_de_log_mai_bun = "Warning.txt"
1235. cls.fileWarning = open(fisier_de_log_mai_bun, "w")

```

```

1236. def handleError(self):
1237. pass
1238. @classmethod
1239. def inchidereFisierCaEAcademic(cls):
1240. cls.file.close()
1241. cls.fileGrav.close()
1242. cls.fileWarning.close()
1243.
1244. class StrategieEroare(Strategie):
1245. def handleError(self):
1246. print("A aparut o eroare!")
1247. self.file.write("Chiar a aparut o eroare!\n")
1248.
1249. class StrategieEroareGrava(Strategie):
1250. def handleError(self):
1251. self.fileGrav.write("Chiar a aparut o eroare grava!\n")
1252. Strategie.inchidereFisierCaEAcademic()
1253. sys.exit()
1254.
1255. class StrategieWarning(Strategie):
1256. def handleError(self):
1257. self.fileWarning.write("Ai picat la PC ca ai warning!\n")
1258.
1259.
1260. class Handler:
1261. def __init__(self):
1262. self.strategie = Strategie()
1263. Strategie.fisiereDeschidere()
1264. def setareStrategie(self, strategie):
1265. self.strategie = strategie
1266. def gestionam(self):
1267. self.strategie.handleError()
1268.
1269. def main():
1270. handler = Handler()
1271. while True:
1272. print("Ce eroare vrei?")
1273. print(" 0 - Grava")
1274. print(" 1 - Normala")
1275. print(" 2 - Doar warning")
1276. opt = int(input(""))
1277. try:
1278. if opt == 0:
1279. raise GraveError
1280. elif opt == 1:
1281. raise Error
1282. else:
1283. raise Warning
1284. except GraveError:
1285. handler.setareStrategie(StrategieEroareGrava())
1286. handler.gestionam()
1287. except Error:
1288. handler.setareStrategie(StrategieEroare())
1289. handler.gestionam()
1290. except Warning:
1291. handler.setareStrategie(StrategieWarning())
1292. handler.gestionam()
1293.
1294.
1295. if __name__ == '__main__':
1296. main()
1297. #sv46 py339

```

strategie_bancuri

Varianta 1

```

1298. from random import choice
1299.
1300.
1301. class Dictionary:
1302. def __init__(self, file_name):
1303. self.joke_book = {
1304.     "": [],
1305.     "***": [],
1306.     "****": []
1307. }
1308. with open(file_name, "r") as file:
1309. for line in file.readlines():
1310. self.joke_book[line.split()[0]].append(line.strip())
1311.
1312. def get_joke(self, joke_rank):
1313. return choice(self.joke_book[joke_rank])
1314.
1315.

```



```

1316. class GoodJokeStrategy:
1317. def __init__(self, joke_collection):
1318. self.joke_collection = joke_collection
1319.
1320. def get_joke(self):
1321. return self.joke_collection.get_joke("")
1322.
1323.
1324. class VeryGoodJokeStrategy:
1325. def __init__(self, joke_collection):
1326. self.joke_collection = joke_collection
1327.
1328. def get_joke(self):
1329. return self.joke_collection.get_joke("****")
1330.
1331.
1332. class TheBestJokeStrategy:
1333. def __init__(self, joke_collection):
1334. self.joke_collection = joke_collection
1335.
1336. def get_joke(self):
1337. return self.joke_collection.get_joke("****")
1338.
1339.
1340. class Student:
1341. def __init__(self, joke_book):
1342. self.joke_book = joke_book
1343. self.strategy = GoodJokeStrategy(self.joke_book)
1344.
1345. def get_feedback(self, feedback):
1346. if feedback == "sad":
1347. self.strategy = TheBestJokeStrategy(self.joke_book)
1348. if feedback == "serious":
1349. self.strategy = VeryGoodJokeStrategy(self.joke_book)
1350. if feedback == "happy":
1351. self.strategy = GoodJokeStrategy(self.joke_book)
1352.
1353. def tell_a_joke(self):
1354. return self.strategy.get_joke()
1355.
1356. def listen(self, joke):
1357. print(f"i listen to {joke}")
1358. joke_rank = joke.split()[0]
1359.
1360. if joke_rank == "":
1361. return "sad"
1362. if joke_rank == "****":
1363. return "serious"
1364. if joke_rank == "****":
1365. return "happy"
1366.
1367.
1368. def main():
1369. joke_book = Dictionary("jokes.txt")
1370. student1 = Student(joke_book)
1371. student2 = Student(joke_book)
1372.
1373. feedback = student2.listen(student1.tell_a_joke())
1374. print(feedback)
1375. student1.get_feedback(feedback)
1376. feedback = student2.listen(student1.tell_a_joke())
1377. student1.get_feedback(feedback)
1378. print(feedback)
1379.
1380.
1381. main()
1382.
1383.
1384.

```

Varianta2

```

1385. import abc
1386. import random
1387. #Se scrie un program Python care utilizand modelul strategii va alege in functie de reactia
1388. #unui coleg un banc dintr-un dictionar care are seturi de vancuri bune, foarte bune si cele mai
1389. #bune ( un fisier text care are un banc pe paragraf si la inceput are una/doua/trei stelute
1390. #(codificarea calitatii vancului). Deci este nevoie de o clasa student, o metoda de a spune
1391. #bancuri si una de a asculta si furniza o reactie.
1392.
1393. def citimGlume(text):#metoda de a spune bancuri
1394. bancuriBune = []
1395. bancuriFoarteBune = []
1396. bancuriCeleMaiBune = []
1397.
1398. default_data = { 'ok': bancuriBune,
1399. 'bune': bancuriFoarteBune,
1400. 'foarte bune':bancuriCeleMaiBune

```

```

1401. }
1402.
1403. glume = text.split("\t")#impartim prin TAB-uri
1404. for element in glume:
1405.     if "****" in element:
1406.         bancuriCeleMaiBune.append( element)#adaugam in vectorul din dictionar
1407.     elif "***" in element:
1408.         bancuriFoarteBune.append( element)
1409.     elif "" in element:
1410.         bancuriBune.append(element)
1411.
1412.
1413. #print(default_data['ok'])
1414. #print(default_data['bune'])
1415. #print(default_data['foarte bune'])
1416. return default_data
1417.
1418. class StrategieGlume( metaclass= abc.ABCMeta):#Strategy
1419.     @abc.abstractmethod
1420.     def spuneGluma(self):
1421.         pass
1422.
1423. class StrategieGlumeOk(StrategieGlume):#ConcreteStrategy
1424.     def __init__(self, _dictionar):
1425.         self.dictionar = _dictionar
1426.
1427.     def spuneGluma(self):
1428.         listas = self.dictionar['ok']
1429.         gluma = listas[random.randint(0, len(listas)-1)]
1430.         print(gluma)
1431.         return gluma
1432.
1433. class StrategieGlumeBune(StrategieGlume):#ConcreteStrategy
1434.     def __init__(self, _dictionar):
1435.         self.dictionar = _dictionar
1436.
1437.     def spuneGluma(self):
1438.         listas = self.dictionar['bune']
1439.         gluma = listas[random.randint(0, len(listas) - 1)]#alegem random o gluma
1440.         print(gluma)#si o listam
1441.         return gluma
1442.
1443. class StrategieGlumeFoarteBune(StrategieGlume):#ConcreteStrategy
1444.     def __init__(self, _dictionar):
1445.         self.dictionar = _dictionar
1446.
1447.     def spuneGluma(self):
1448.         listas = self.dictionar['foarte bune']
1449.         gluma = listas[random.randint(0,len(listas) -1) ]
1450.         print( gluma)
1451.         return gluma
1452.
1453. class Student():#0 clasa context
1454.     def __init__(self, _strategie):
1455.         self.strategie = _strategie
1456.
1457.     def spuneGluma(self):
1458.         return self.strategie.spuneGluma()#apeleaza metoda din Clasele ce extind StrategieGlume
1459.
1460.     def ascultaGluma(self,gluma):
1461.         if "****" in gluma:
1462.             print("hahaHAHAHAHAHAHA")#dupa ce se spune gluma,se asculta
1463.         elif "***" in gluma:
1464.             print("hahaHAHA")
1465.         elif "" in gluma:
1466.             print("ha")
1467.
1468.     if __name__ == '__main__':
1469.         f = open("bancuri.txt", "r")
1470.         text = f.read()
1471.
1472.         dictionarGlume = citimGlume(text)
1473.         #print(glume)
1474.         strategie1 = StrategieGlumeOk(dictionarGlume);
1475.         strategie2 = StrategieGlumeFoarteBune(dictionarGlume);
1476.
1477.         student1 = Student(strategie1)
1478.         student2 = Student(strategie2)
1479.
1480.         student2.ascultaGluma(student1.spuneGluma())
1481.         student1.ascultaGluma(student2.spuneGluma())

```

threading_intretesarea_linieCulinie

```
1482. import threading
1483. from functional import seq
1484.
1485. #intretesare linii din fisiere in adt
1486. #prelucrare linii din adt
1487.
1488. class Tailor(threading.Thread):
1489. def __init__(self, file1, file2, file3, file4):
1490. threading.Thread.__init__(self)
1491. self.file1=file1
1492. self.file2=file2
1493. self.file3=file3
1494. self.file4=file4
1495. self.lines=[]
1496. def sew_lines(self):
1497. with open(self.file1) as fis1:
1498. with open(self.file2) as fis2:
1499. with open(self.file3) as fis3:
1500. with open(self.file4) as fis4:
1501.
1502. carry_on=True
1503. while carry_on:
1504. carry_on=False
1505.
1506. line1=fis1.readline()
1507. if len(line1) >0:
1508. self.lines.append(line1)
1509. carry_on=True
1510.
1511. for _ in range(0,2):
1512. line2=fis2.readline()
1513. if len(line2) >0:
1514. self.lines.append(line2)
1515. carry_on=True
1516.
1517. for _ in range(0,3):
1518. line3=fis3.readline()
1519. if len(line3) >0:
1520. self.lines.append(line3)
1521. carry_on=True
1522.
1523. for _ in range(0,4):
1524. line4=fis4.readline()
1525. if len(line4) >0:
1526. self.lines.append(line4)
1527. carry_on=True
1528.
1529. self.lines=seq(self.lines).map(func=lambda line: line.strip())
1530.
1531. with open("result.txt","w") as result:
1532. for line in self.lines:
1533. result.write(line)
1534.
1535. def run(self):
1536. self.sew_lines()
1537.
1538. def main():
1539. tailor=Tailor("fis1.txt", "fis2.txt", "fis3.txt", "fis4.txt")
1540. tailor.start()
1541. main()
1542. #simulare
```

tkinter_lambda_mouse

```
from tkinter import *
1543. from queue import LifoQueue
1544.
1545. class Drawer(Frame):
1546. def __init__(self, root, **kw):
1547. super().__init__(**kw)
1548. self.root = root
1549. self.stack = LifoQueue()
1550. self.canvas = Canvas(root, width=900, height=900)
1551. self.canvas.grid(row=1, column=0)
1552. self.canvas.bind('<Button-1>', self.addPoint)
1553. self.label = Label(self.canvas, text="Label lol")
1554. self.label.place(relx=0.5, rely=0.05, anchor=CENTER)
1555. self.buttonQuit = Button(self.root, text="Quit", padx=20, pady=20, command=self.quit).grid(row=0, column=0)
1556. self.buttonClear = Button(self.root, text="Clear", padx=20, pady=20, command=self.clear).grid(row=0, column=1)
1557. #self.canvas.pack()
1558. def addPoint(self, event):
```

```

1559. self.label.configure(text="Am pus un punct la " + str(event.x) + " si " + str(event.y))
1560. if not self.stack.empty():
1561. point1 = self.stack.get()
1562. self.canvas.create_line(point1[0], point1[1], event.x, event.y)
1563. #self.canvas.pack()
1564. self.stack.put((event.x, event.y))
1565. def quit(self):
1566. import sys
1567. sys.exit()
1568. def clear(self):
1569. self.label.configure(text="Am sters tot!")
1570. self.canvas.delete("all")
1571. #self.canvas.pack()
1572. while not self.stack.empty():
1573. self.stack.get()
1574.
1575. if __name__ == '__main__':
1576. root = Tk()
1577. root.title("cod maimuta")
1578. drawer = Drawer(root)
1579. root.mainloop()
1580. sv38py346
1581.
1582.
1583.
1584.
1585.
1586.
1587.
1588.
1589.
1590.
1591.
1592.
1593.

```

KOTLIN

2CARACTERE_DIN_MIJLOC_CUVANT_STERGERE

```

1594. import java.io.File
1595.
1596. fun main(args:Array<String>) {
1597. var a=""
1598. File("in.txt").forEachLine { a+=it
1599. a+=" "
1600. }
1601. val words:List<String>
1602. words = a.split(" ")
1603.
1604. val new_words=words.filter { it.length >= 4 }
1605. .map { it.removeRange(it.length/2-1,it.length/2+1)}
1606.
1607. println(new_words)
1608.
1609. }
1610.
1611. ///Stergere sv34 Kot40
1612. import java.io.File
1613.
1614. fun main(args:Array<String>) {
1615. var a=""
1616. File("out.txt").forEachLine { a+=it
1617. a+=" "
1618. }
1619. val words:List<String>
1620. words = a.split(" ")
1621.
1622. val new_words=words.filter { it.length >= 4 }
1623. .map { it.removeRange(0,2)}
1624.
1625. println(new_words)
1626.
1627. }

```

2colectii,15nr,aleator,AxB

```
1628. fun main(args:Array<String>) {
1629.
1630. val A = mutableListOf<Int>()
1631. val B = mutableListOf<Int>()
1632.
1633. for (i in 0..15) {
1634. A.add((0..100).random())
1635. B.add((0..100).random())
1636. }
1637.
1638. println(A)
1639. println(B)
1640.
1641. var AxB = A.map { x->B.map { y->Pair(x,y) } }
1642. println(AxB)
1643. //sv44 kot57
1644.
1645.
1646.
```

2_colectii_100nr_funcctie_multimi

```
1647. val computere={
1648. A:Set<Int>,B:Set<Int> ->
1649. var X = setOf<Pair<Int,Int>>()
1650. for (i in A){
1651. for (j in B)
1652. {
1653. var p=Pair(i,j)
1654. X+=p
1655. }
1656. }
1657. X
1658. }
1659. val intersectie={
1660. A:Set<Int>,B:Set<Int> ->
1661. var X = setOf<Int>()
1662. for (i in A){
1663. for (j in B)
1664. {
1665. if(i==j)
1666. X+=i
1667. }
1668. }
1669. X
1670. }
1671. val reuniune={
1672. A:Set<Pair<Int,Int>>,B:Set<Int> ->
1673. var X = hashMapOf<Int,Int>()
1674. for (i in A){
1675. X+=i
1676. }
1677. for(j in B)
1678. {
1679. X+=Pair(j,0)
1680. }
1681. X
1682. }
1683. fun main(){
1684. var A = mutableSetOf<Int>()
1685. var B = mutableSetOf<Int>()
1686. var countA=0
1687. var countB=0
1688. var i=0
1689. while(countA<100)
1690. {
1691. var numator=8*i-18
1692. var numitor=2*i-9
1693. if(numator/numitor >=0)
1694. {
1695. A+=numator/numitor
1696. countA+=1
1697. }
1698. i+=1
1699. println(countA)
1700. }
1701. i=0
1702. while(countB<100)
1703. {
1704. var numator=9*i*i -48*i +16
1705. var numitor=3*i-8
1706. if(numator/numitor >=0)
1707. {
```

```

1708. B+=numerator/numitor
1709. countB+=1
1710. }
1711. i+=1
1712. println(countB)
1713. }
1714. println("A="+A)
1715. println("B="+B)
1716. /*var AxB=comunere(A,B)
1717. println("(AxB)="+AxB)
1718. var BxA=intersectie(B,A)
1719. println("(BxA)="+BxA)
1720. var comun=reuniune(AxB,BxA)
1721. println("(AxB) u (BxA)="+comun)*/
1722.
1723. var AxB = A.map { x->B.map { y->Pair(x,y) } }
1724. // println(AxB)
1725.
1726. var AXB= mutableListOf<Pair<Int, Int>>()
1727. AxB.forEach { it.forEach { AXB.add(it) } }
1728. println(AXB)
1729.
1730. var BnA= B.filter { A.contains(it) }
1731. println(BnA)
1732.
1733. var result = AXB+BnA
1734. println(result)
1735. }
1736. //sv53 kot65
1737.
1738.

```

2_colectii_100nr_functie_multimi #varianta2

```

1739. fun main(args:Array<String>) {
1740.
1741. val A = mutableListOf<Int>()
1742. val B = mutableListOf<Int>()
1743.
1744. for (i in 0..99)
1745. A.add((8*i-18)/(2*i-9))
1746. for (i in 0..99)
1747. B.add((9*i*i-48*i+16)/(3*i-8))
1748.
1749. println(A)
1750. println(B)
1751.
1752. var AxB = A.map { x->B.map { y->Pair(x,y) } }
1753. // println(AxB)
1754.
1755. var AXB= mutableListOf<Pair<Int, Int>>()
1756. AxB.forEach { it.forEach { AXB.add(it) } }
1757. println(AXB)
1758.
1759. var BnA= B.filter { A.contains(it) }
1760. println(BnA)
1761.
1762. var result = AXB+BnA
1763. println(result)
1764.
1765.
1766. ///SAU
1767. import kotlin.math.floor
1768.
1769. fun generatecollection1(): MutableList<Float> {
1770.
1771.
1772. val coll = mutableListOf<Float>()
1773.
1774. var index = 0
1775. var n: Int = 0
1776. var x: Float
1777.
1778. while (index < 100 && n < 1000)
1779. {
1780. x = ((8.0 * n - 18.0) / (2.0 * n - 9.0)).toFloat()
1781. if(floor(x) == x && x > 0) {
1782. coll.add(x)
1783. //println("chestie gasita [$index] [$n] [$x]")
1784. index++
1785. }
1786. n++
1787. }
1788. return coll
1789. }
1790.
1791.
1792. fun generatecollection2():MutableList<Float>
1793. {

```

```

1794. val coll = mutableList<Float>()
1795.
1796. var index = 0
1797. var n: Int = 0
1798. var x: Float
1799.
1800. while (index < 100 && n < 1000)
1801. {
1802. x = ((9.0 * n * n - 48.0 * n + 16.0) / (3.0 * n - 8.0)).toFloat()
1803. if(floor(x) == x ) {
1804.
1805. coll.add(x)
1806. //println("chestie gasita [$index] [$n] [$x]")
1807. index++
1808.
1809. }
1810. n++
1811. }
1812. return coll
1813. }
1814.
1815. fun main() {
1816. var collection1:List<Float> = generatecollection1()
1817. collection1 = collection1.sorted()
1818. println("Colectia 1: $collection1")
1819.
1820. var collection2:List<Float> = generatecollection2()
1821. collection2 = collection2.sorted()
1822. println("Colectia 2: $collection2")
1823.
1824. var hashmap = (1..listOf(collection1.map { c1-> collection2.map { c2-> Pair(c1,c2)}}.flatten(),
    collection1.intersect(collection2)).flatten().size).zip(
1825. listOf(collection1.map { c1-> collection2.map { c2-> Pair(c1,c2)}}.flatten(), collection1.intersect(collection2)).flatten()).toMap()
1826.
1827. print(hashmap)
1828.
1829. }
1830. sv53 kot65
1831.

```

AB_AxB_lambda

```

1832. import java.util.Random
1833.
1834. val computere={
1835. A:Set<Int>,B:Set<Int> ->
1836. var X = setOf<Pair<Int,Int>>()
1837. for (i in A){
1838. for (j in B)
1839. {
1840. var p=Pair(i,j)
1841. X+=p
1842. }
1843. }
1844. X
1845. }
1846.
1847. fun main()
1848. {
1849. var A = setOf<Int>()
1850. var B = setOf<Int>()
1851. for (i in 1..16)
1852. {
1853. A+= (0..999).random()
1854. B+= (0..999).random()
1855. }
1856. var X=computere(A,B)
1857. println("AxB="+X)
1858.
1859. var AxB = A.map { x->B.map { y->Pair(x,y) } }
1860. println(AxB)
1861. }
1862. //sv44kot57

```

adaptor_datecolectii_afisareca_operatie

```

1863. import java.io.File
1864.
1865. interface NUMBER
1866. {
1867. fun afiseazaDouble(fw:File)
1868. fun afiseazaFloat(fw:File)
1869. fun afiseazaInt(fw:File)
1870. }
1871. class DOUBLE(val nr: Number):NUMBER

```

```

1872. {
1873. override fun afiseazaDouble(fw:File) {
1874. fw.appendText("Am afisat un double: " + nr + "\n")
1875. }
1876. override fun afiseazaFloat(fw:File) {
1877. }
1878. override fun afiseazaInt(fw:File) {
1879. }
1880. override fun toString(): String {
1881. return "double"
1882. }
1883. }
1884. class INTEGER(val nr: Number):NUMBER
1885. {
1886. override fun afiseazaDouble(fw:File) {
1887. }
1888. override fun afiseazaFloat(fw:File) {
1889. }
1890. override fun afiseazaInt(fw:File) {
1891. fw.appendText("Am afisat un intreg: " + nr + "\n")
1892. }
1893. override fun toString(): String {
1894. return "int"
1895. }
1896. }
1897.
1898. class FLOAT(val nr: Number):NUMBER
1899. {
1900. override fun afiseazaDouble(fw:File) {
1901. }
1902.
1903. override fun afiseazaFloat(fw:File) {
1904. fw.appendText("Am afisat un float: " + nr + "\n")
1905. }
1906.
1907. override fun afiseazaInt(fw:File) {
1908. }
1909. }
1910. override fun toString(): String {
1911. return "float"
1912. }
1913. }
1914.
1915. class Complex(val re:Number, val im:Number)
1916. {
1917. fun afiseazaComplex(fw: File)
1918. {
1919. fw.appendText("Am afisat un complex: " + re + " " +im+"*i" + "\n")
1920. }
1921. override fun toString(): String {
1922. return "complex"
1923. }
1924. }
1925. interface FileWriter
1926. {
1927. fun Write(fw: File, obj: Any)
1928. }
1929.
1930. open class Adapter:FileWriter
1931. {
1932. override fun Write(fw:File, obj: Any) {
1933. val aux: NUMBER = obj as NUMBER
1934. when(aux.toString())
1935. {
1936. "double"-> aux.afiseazaDouble(fw)
1937. "int"-> aux.afiseazaInt(fw)
1938. "float"-> aux.afiseazaFloat(fw)
1939. else -> fw.appendText("numar incorect" + "\n")
1940. }
1941. }
1942. }
1943.
1944. class Executor:Adapter()
1945. {
1946. override fun Write(fw: File, obj: Any) {
1947. if (obj.toString() == "double" || obj.toString() == "int" || obj.toString() == "float")
1948. {
1949. super.Write(fw, obj)
1950. }
1951. else
1952. if(obj.toString() == "complex")
1953. {
1954. val aux: Complex = obj as Complex
1955. aux.afiseazaComplex(fw)
1956. }
1957. }
1958. }
1959. fun main(args:Array<String>)
1960. {
1961. val file = File("out.txt")
1962. val executor = Executor()
1963.

```



```

1964. val intreg = INTEGER(700)
1965. val real = DOUBLE(800.6)
1966. val floatNr = FLOAT(32.6f)
1967. val complex = Complex(10,80)
1968.
1969. executor.Write(file,intreg)
1970. executor.Write(file,real)
1971. executor.Write(file,floatNr)
1972. executor.Write(file,complex)
1973. }
1974. //sv81 kot71

```

adaptor_datecolectii_afisare_ca_o_singura_operatie

```

1975. import java.io.File
1976.
1977. interface NUMBER
1978. {
1979. fun afiseazaDouble()
1980. fun afiseazaFloat()
1981. fun afiseazaInt()
1982. }
1983. class DOUBLE(val nr: Number):NUMBER
1984. {
1985. override fun afiseazaDouble() {
1986. println("Am afisat un double: " + nr + "\n")
1987. }
1988. override fun afiseazaFloat() {
1989. }
1990. override fun afiseazaInt() {
1991. }
1992. override fun toString(): String {
1993. return "double"
1994. }
1995. }
1996. class INTEGER(val nr: Number):NUMBER
1997. {
1998. override fun afiseazaDouble() {
1999. }
2000. override fun afiseazaFloat() {
2001. }
2002. override fun afiseazaInt() {
2003. println("Am afisat un intreg: " + nr + "\n")
2004. }
2005. override fun toString(): String {
2006. return "int"
2007. }
2008. }
2009.
2010. class FLOAT(val nr: Number):NUMBER
2011. {
2012. override fun afiseazaDouble() {
2013. }
2014.
2015. override fun afiseazaFloat() {
2016. println("Am afisat un float: " + nr + "\n")
2017. }
2018.
2019. override fun afiseazaInt() {
2020. }
2021. }
2022. override fun toString(): String {
2023. return "float"
2024. }
2025. }
2026.
2027. class Complex(val re:Number, val im:Number)
2028. {
2029. fun afiseazaComplex()
2030. {
2031. println("Am afisat un complex: " + re + " " +im+"*i" + "\n")
2032. }
2033. override fun toString(): String {
2034. return "complex"
2035. }
2036. }
2037. interface Writer
2038. {
2039. fun Write(obj: Any)
2040. }
2041.
2042. open class Adapter:Writer
2043. {
2044. override fun Write(obj: Any) {
2045. val aux: NUMBER = obj as NUMBER
2046. when(aux.toString())

```

```

2047. {
2048. "double"-> aux.afiseazaDouble()
2049. "int"-> aux.afiseazaInt()
2050. "float"-> aux.afiseazaFloat()
2051. else -> println("numar incorect" + "\n")
2052. }
2053. }
2054. }
2055.
2056. class Executor:Adapter()
2057. {
2058. override fun Write(obj: Any) {
2059. if (obj.toString() == "double" || obj.toString() == "int" || obj.toString() == "float")
2060. {
2061. super.Write(obj)
2062. }
2063. else
2064. if(obj.toString() == "complex")
2065. {
2066. val aux: Complex = obj as Complex
2067. aux.afiseazaComplex()
2068. }
2069. }
2070. }
2071. fun main(args:Array<String>) {
2072. val file = File("out.txt")
2073. val executor = Executor()
2074.
2075. val intreg = INTEGER(700)
2076. val real = DOUBLE(800.6)
2077. val floatNr = FLOAT(32.6f)
2078. val complex = Complex(10, 80)
2079.
2080. executor.Write(intreg)
2081. executor.Write(real)
2082. executor.Write(floatNr)
2083. executor.Write(complex)
2084. }
2085. //sv80 kot70

```

ad_t_lambda_nr_submultimi_4el

```

2086. val procesareSubmultimi={
2087. A:Set<Int> ->
2088. var array:MutableSet<MutableSet<Int>> = mutableSetOf()
2089. subset(array, mutableSetOf<Int>(),A,0)
2090. array
2091. }
2092. fun subset(S: MutableSet<MutableSet<Int>> , resultSet:MutableSet<Int>, A: Set<Int>,start: Int)
2093. {
2094. S.add(mutableSetOf<Int>())
2095. for(i in start..A.size-1)
2096. {
2097. resultSet.add(A.elementAt(i))
2098. subset(S,resultSet,A,i+1)
2099. resultSet.remove(resultSet.size - 1)
2100. }
2101.
2102. }
2103.
2104. fun main()
2105. {
2106. val A=1.until(101).toSet()
2107. var count = 0
2108. val subsets=procesareSubmultimi(A)
2109. print(subsets)
2110. subsets.forEach {
2111. if(it.size == 4 && it.contains(1))
2112. count+=1
2113. }
2114. println("Numarul submultimilor cu 4 elemente si care contin elementul 1:"+count)
2115. }
2116. //sv38kot52

```

arbore_binar_functor_gasire_parsefile

```

2117. import java.io.File
2118.
2119. class ArboreBinar(var cuvant: String) {
2120. var left: ArboreBinar? = null
2121. var right: ArboreBinar? = null
2122. }

```

```

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

```
2207. val comunere={
2208. A:Set<Int>,B:Set<Int> ->
2209. var X = hashMapOf<Int,Int>()
2210. for (i in A){
2211. for (j in B)
2212. {
2213. var p=Pair(i,j)
2214. if(!X.containsKey(i) || !X.containsValue(j))
2215. X+=p
2216. }
2217. }
2218. X
2219. }
2220. val intersectie={
2221. A:Set<Int>,B:Set<Int> ->
2222. var X = setOf<Int>()
2223. for (i in A){
2224. for (j in B)
2225. {
2226. if(i==j)
2227. X+=i
2228. }
2229. }
2230. X
2231. }
2232. val reuniune={
2233. A:Set<Int>,B:Set<Int> ->
2234. var X = setOf<Int>()
2235. for (i in A){
2236. X+=i
2237. }
2238. for(j in B)
2239. {
2240. if(!X.contains(j))
2241. X+=j
2242. }
2243. X
2244. }
2245. fun main()
2246. {
2247. var A = mutableSetOf<Int>()
2248. var B = mutableSetOf<Int>()
2249. for (i in 1..21)
2250. {
2251. A+= (0..5).random()
2252. B+= (0..5).random()
2253. }
2254. println("A="+A)
2255. println("B="+B)
2256. /*var AuB=reuniune(A,B)
2257. println("(AuB)="+AuB)
2258. var BnA=intersectie(B,A)
2259. println("(BnA)="+BnA)
2260. var comun=comunere(AuB,BnA)
2261. println("(AuB) x (BnA)="+comun)*/
2262.
2263. val AuB = A.filter { !B.contains(it) } + B
2264. println(AuB)
2265.
2266. val AnB =A.filter { B.contains(it) }
2267. println(AnB)
2268.
2269. var AuBxBnA = AuB.map { x->AnB.map { y->Pair(x,y) } }
2270. println(AuBxBnA)
2271.
2272. var AUBXBNA= mutableListOf<Pair<Int, Int>>()
2273. AuBxBnA.forEach { it.forEach { AUBXBNA.add(it) } }
2274. println(AUBXBNA.toMap())
2275. }
2276.
2277. SAU
2278. ///
2279. fun main(args:Array<String>) {
2280.
2281. val A = mutableListOf<Int>()
2282. val B = mutableListOf<Int>()
2283.
2284. for (i in 0..5)
2285. A.add(i)
2286. for (i in 3..8)
2287. B.add(i)
2288.
2289. println(A)
2290. println(B)
2291.
```

```

2292. val AuB = A.filter { !B.contains(it) } + B
2293. println(AuB)
2294.
2295. val AnB =A.filter { B.contains(it) }
2296. println(AnB)
2297.
2298. var AuBxBnA = AuB.map { x->AnB.map { y->Pair(x,y) } }
2299. println(AuBxBnA)
2300.
2301. var AUBXBNA= mutableListOf<Pair<Int, Int>>()
2302. AuBxBnA.forEach { it.forEach { AUBXBNA.add(it) } }
2303. println(AUBXBNA.toMap())
2304.
2305. sv50 kot62

```

--- automat_inchidere_paranteze ---

```

2306. import java.io.BufferedReader
2307. import java.io.File
2308.
2309. class Automat
2310. {
2311.     var state: State
2312.     var par_deschise: Int = 0
2313.     init{
2314.         state = ParantezeState(this, "C:\\Users\\monic\\OneDrive\\Desktop\\kotlinpp\\kotlinSource.txt")
2315.     }
2316.     fun verificaInchidereCorecta(s: String): Boolean = state.verificaInchidereCorecta(s)
2317.     fun extrageParanteze(): String = state.extrageParanteze()
2318. }
2319.
2320. abstract class State(val automat: Automat, val filename: String)
2321. {
2322.     abstract fun extrageParanteze(): String
2323. }
2324. abstract fun verificaInchidereCorecta(s: String): Boolean
2325.
2326. fun eParantezaDeschisa(c: Char): Boolean
2327. {
2328.     if(c == '(' || c == '[' || c == '{')
2329.         return true
2330.     return false
2331. }
2332.
2333. fun eParantezaInchisa(c: Char): Boolean
2334. {
2335.     if(c == ')' || c == ']' || c == '}')
2336.         return true
2337.     return false
2338. }
2339. }
2340.
2341. class ParantezeState(automat: Automat, filename: String): State(automat,filename)
2342. {
2343.     override fun extrageParanteze(): String
2344.     {
2345.         var result :String = ""
2346.         val bufferedReader = File(filename).bufferedReader()
2347.         val inputString = bufferedReader.use { it.readText() }
2348.         var i = 0
2349.         while(i < inputString.length)
2350.         {
2351.             if(eParantezaDeschisa(inputString[i])) {
2352.                 automat.par_deschise += 1
2353.                 result += inputString[i]
2354.             }
2355.             if (eParantezaInchisa(inputString[i])) {
2356.                 automat.par_deschise -= 1
2357.                 result += inputString[i]
2358.             }
2359.             i += 1
2360.         }
2361.         return result
2362.     }
2363.
2364.     override fun verificaInchidereCorecta(s: String): Boolean
2365.     {
2366.         if(automat.par_deschise == 0)
2367.             return true
2368.         return false
2369.     }
2370. }
2371.
2372.
2373. fun main(args: Array<String>){
2374. {
2375.     val automat = Automat()
2376.     var paranteze = automat.extrageParanteze()

```

```

2377. println(paranteze)
2378.
2379. var verifica = automat.verificaInchidereCorecta(paranteze)
2380.
2381. if(verifica == true)
2382. println("Parantezele au fost inchise corect!")
2383. else
2384. println("Parantezele nu au fost inchise corect!")
2385. }

```

AxB U BxB_dictionar

```

2386. val computere={
2387. A:Set<Int>,B:Set<Int> ->
2388. var X = setOf<Pair<Int,Int>>()
2389. for (i in A){
2390. for (j in B)
2391. {
2392. var p=Pair(i,j)
2393. X+=p
2394. }
2395. }
2396. X
2397. }
2398. val reuniune={
2399. A:Set<Pair<Int,Int>>,B:Set<Pair<Int,Int>> ->
2400. var X = hashMapOf<Int,Int>()
2401. for (i in A){
2402. X+=i
2403. }
2404. for(j in B)
2405. {
2406. if(!X.containsKey(j.first) || !X.containsValue(j.second))
2407. X+=j
2408. }
2409. X
2410. }
2411. fun main()
2412. {
2413. var A = mutableSetOf<Int>()
2414. var B = mutableSetOf<Int>()
2415. for (i in 1..21)
2416. {
2417. A+= (0..5).random()
2418. B+= (0..5).random()
2419. }
2420. println("A="+A)
2421. println("B="+B)
2422. /*var AxB=computere(A,B)
2423. println("(AxB)="+AxB)
2424. var BxB=computere(B,B)
2425. println("(BxB)="+BxB)
2426. var comun=reuniune(AxB,BxB)
2427. println("(AxB) u (BxB)="+comun)*/
2428.
2429. var AxB = A.map { x->B.map { y->Pair(x,y) } }
2430. // println(AxB)
2431.
2432. var AXB= mutableListOf<Pair<Int, Int>>()
2433. AxB.forEach { it.forEach { AXB.add(it) } }
2434. println(AXB)
2435.
2436. var BxB = B.map { x->B.map { y->Pair(x,y) } }
2437. //println(BxB)
2438.
2439. var BXB= mutableListOf<Pair<Int, Int>>()
2440. BxB.forEach { it.forEach { BXB.add(it) } }
2441. println(BXB)
2442.
2443. var AxBuBxB= AXB.filter { !BXB.contains(it) }
2444. AxBuBxB+=(BXB.filter { BXB.contains(it) })
2445. println(AxBuBxB)
2446.
2447. println(AxBuBxB.toMap())
2448. }
2449. SAU
2450. ////////////
2451.
2452. fun main(args:Array<String>) {
2453.
2454. val A = mutableListOf<Int>()
2455. val B = mutableListOf<Int>()
2456.
2457. for (i in 0..2)
2458. A.add(i)
2459. for (i in 1..3)
2460. B.add(i)

```

```

2461.
2462. println(A)
2463. println(B)
2464.
2465. var AxB = A.map { x->B.map { y->Pair(x,y) } }
2466. println(AxB)
2467.
2468. var BxA = B.map { x->A.map { y->Pair(x,y) } }
2469. println(BxA)
2470.
2471. var AXB= mutableListOf<Pair<Int, Int>>()
2472. AxB.forEach { it.forEach { AXB.add(it) } }
2473.
2474. var BXA= mutableListOf<Pair<Int, Int>>()
2475. BxA.forEach { it.forEach { BXA.add(it) } }
2476.
2477. println(AXB)
2478. println(BXA)
2479.
2480. val result = AXB.filter { BXA.contains(it) }.toMap()
2481. println(result)
2482.
2483. }
2484.
2485. //sv47 kot60

```

AxB x BuA_dictionar

```

2486. fun main(args:Array<String>) {
2487.
2488. val A = mutableListOf<Int>()
2489. val B = mutableListOf<Int>()
2490.
2491. for (i in 0..2)
2492. A.add(i)
2493. for (i in 1..3)
2494. B.add(i)
2495.
2496. println(A)
2497. println(B)
2498.
2499. var AxB = A.map { x->B.map { y->Pair(x,y) } }
2500. println(AxB)
2501.
2502. var AXB= mutableListOf<Pair<Int, Int>>()
2503. AxB.forEach { it.forEach { AXB.add(it) } }
2504. println(AXB)
2505.
2506. var BuA= A.filter { !B.contains(it) }
2507. BuA+=(B.filter { B.contains(it) })
2508. println(BuA)
2509.
2510. var AxBxBuA = AXB.map { x->B.map { y->Pair(x,y) } }
2511. println(AxBxBuA)
2512.
2513. var AXBxBuA= mutableListOf<Pair<Pair<Int, Int>,Int>>()
2514. AxBxBuA.forEach { it.forEach { AXBxBuA.add(it) } }
2515.
2516. println(AXBxBuA.toMap())
2517.
2518.
2519. }
2520.
2521. //////////////sau
2522. val reuniune={
2523. A:Set<Int>,B:Set<Int> ->
2524. var X = setOf<Int>()
2525. for (i in A){
2526. X+=i
2527. }
2528. for(j in B)
2529. {
2530. if(!X.contains(j))
2531. X+=j
2532. }
2533. X
2534. }
2535. val computere={
2536. A:Set<Int>,B:Set<Int> ->
2537. var X = setOf<Pair<Int,Int>>()
2538. for (i in A){
2539. for (j in B)
2540. {
2541. var p=Pair(i,j)
2542. X+=p
2543. }

```

```

2544. }
2545. X
2546. }
2547. val comuneref={
2548. A:Set<Pair<Int,Int>>,B:Set<Int> ->
2549. var X = hashMapOf<Pair<Int,Int>,Int>()
2550. for (i in A){
2551. for (j in B)
2552. {
2553. var p=Pair(i,j)
2554. if(!X.containsKey(i) || !X.containsValue(j))
2555. X+=p
2556. }
2557. }
2558. X
2559. }
2560. fun main()
2561. {
2562. var A = mutableSetOf<Int>()
2563. var B = mutableSetOf<Int>()
2564. for (i in 0..2)
2565. A.add(i)
2566. for (i in 1..3)
2567. B.add(i)
2568. println("A="+A)
2569. println("B="+B)
2570. /*var AxB=comunere(A,B)
2571. println("(AxB)="+AxB)
2572. var BxB=reuniune(B,A)
2573. println("(BuA)="+BxB)
2574. var comun=comuneref(AxB,BxB)
2575. println("(AxB) x (BuA)="+comun)*/
2576.
2577. var AxB = A.map { x->B.map { y->Pair(x,y) } }
2578. println(AxB)
2579.
2580. var AXB= mutableListOf<Pair<Int, Int>>()
2581. AxB.forEach { it.forEach { AXB.add(it) } }
2582. println(AXB)
2583.
2584. var BuA= A.filter { !B.contains(it) }
2585. BuA+=(B.filter { B.contains(it) })
2586. println(BuA)
2587.
2588. var AxBxBuA = AXB.map { x->B.map { y->Pair(x,y) } }
2589. println(AxBxBuA)
2590.
2591. var AXBxBUA= mutableListOf<Pair<Pair<Int, Int>,Int>>()
2592. AxBxBuA.forEach { it.forEach { AXBxBUA.add(it) } }
2593.
2594. println(AXBxBUA.toMap())
2595. }
2596. //sv51Kot63

```

AxBintersectBxA_dictionar

```

2597. val comunere={
2598. A:Set<Int>,B:Set<Int> ->
2599. var X = setOf<Pair<Int,Int>>()
2600. for (i in A){
2601. for (j in B)
2602. {
2603. var p=Pair(i,j)
2604. X+=p
2605. }
2606. }
2607. X
2608. }
2609. val intersectie={
2610. A:Set<Pair<Int,Int>>,B:Set<Pair<Int,Int>> ->
2611. var X = hashMapOf<Int,Int>()
2612. for (i in A){
2613. for (j in B)
2614. {
2615. if(i.first==j.first && i.second==j.second)
2616. if(!X.containsKey(i.first) || !X.containsValue(i.second))
2617. X.put(i.first,i.second)
2618. }
2619. }
2620. X
2621. }
2622. fun main()
2623. {
2624. var A = mutableSetOf<Int>()
2625. var B = mutableSetOf<Int>()
2626. for (i in 1..21)
2627. {

```



```

2628. A+= (0..5).random()
2629. B+= (0..5).random()
2630. }
2631. println("A="+A)
2632. println("B="+B)
2633. /*var AxB=compunere(A,B)
2634. println("(AxB)="+AxB)
2635. var BxA=compunere(B,A)
2636. println("(BxA)="+BxA)
2637. var comun=intersectie(AxB,BxA)
2638. println("(AxB) n (BxA)="+comun)*/
2639.
2640. var AxB = A.map { x->B.map { y->Pair(x,y) } }
2641. println(AxB)
2642.
2643. var BxA = B.map { x->A.map { y->Pair(x,y) } }
2644. println(BxA)
2645.
2646. var AXB= mutableListOf<Pair<Int, Int>>()
2647. AxB.forEach { it.forEach { AXB.add(it) } }
2648.
2649. var BXA= mutableListOf<Pair<Int, Int>>()
2650. BxA.forEach { it.forEach { BXA.add(it) } }
2651.
2652. println(AXB)
2653. println(BXA)
2654.
2655. val result = AXB.filter { BXA.contains(it) }.toMap()
2656. println(result)
2657. }
2658.
2659. ///////////
2660. //SAU
2661. fun main(args:Array<String>) {
2662.
2663. val A = mutableListOf<Int>()
2664. val B = mutableListOf<Int>()
2665.
2666. for (i in 0..2)
2667. A.add(i)
2668. for (i in 1..3)
2669. B.add(i)
2670.
2671. println(A)
2672. println(B)
2673.
2674. var AxB = A.map { x->B.map { y->Pair(x,y) } }
2675. println(AxB)
2676.
2677. var BxA = B.map { x->A.map { y->Pair(x,y) } }
2678. println(BxA)
2679.
2680. var AXB= mutableListOf<Pair<Int, Int>>()
2681. AxB.forEach { it.forEach { AXB.add(it) } }
2682.
2683. var BXA= mutableListOf<Pair<Int, Int>>()
2684. BxA.forEach { it.forEach { BXA.add(it) } }
2685.
2686. println(AXB)
2687. println(BXA)
2688.
2689. val result = AXB.filter { BXA.contains(it) }.toMap()
2690. println(result)
2691.
2692. }
2693.
2694. //sv46kot59
2695. -----
2696. corutine_text_kotlin
2697. -----
2698. import kotlinx.coroutines.*
2699. import kotlinx.coroutines.channels.Channel
2700. import kotlinx.coroutines.sync.Mutex
2701. import kotlinx.coroutines.sync.withLock
2702. import java.io.File
2703.
2704. fun main() = runBlocking <Unit>{
2705. val mutex= Mutex()
2706. val channel = Channel<String>(Channel.CONFLATED)
2707. var final : Boolean = false
2708. var which : Int = 1
2709. launch {
2710. var s :String
2711. var s1 : String
2712. val f = File("/home/mihai/git-
server/repos/1211a_zaharia_tedor_mihai.git/Laborator10/Laborator10/src/main/kotlin/Pregatire/Text.txt").bufferedReader()
2713. while(f.ready()) {
2714. s = f.readLine()
2715. while(s.indexOf(" ")!=-1)
2716. {
2717. s1=s.substring(0,s.indexOf(" "))
2718. if(channel.isEmpty) {

```

```

2719. channel.send(s1)
2720. s = s.substring(s.indexOf(" ") + 1, s.length)
2721. }
2722. else
2723. yield()
2724. }
2725. yield()
2726. channel.send(s)
2727. }
2728. final = true
2729. }
2730. launch {
2731. while (!channel.isEmpty||final==false)
2732. if(which==1) {
2733. stringReceive(1, channel)
2734. which=2
2735. }
2736. else{
2737. yield()
2738. }
2739. }
2740. launch {
2741. while (!channel.isEmpty||final==false)
2742. if(which==2) {
2743. stringReceive(2, channel)
2744. which=3
2745. }
2746. else{
2747. yield()
2748. }
2749. }
2750. launch {
2751. while (!channel.isEmpty||final==false)
2752. if(which==3) {
2753. stringReceive(3, channel)
2754. which=1
2755. }
2756. else{
2757. yield()
2758. }
2759. }
2760. }
2761. suspend fun stringReceive(i:Int, c : Channel<String>)
2762. {
2763. var s: String = ""
2764. s=c.receive()
2765. println("Mesajul "+s+" transmis de corutina "+i)
2766. }

```

----- funct_colectii_sumCuMult -----

```

2767. object functie{
2768. fun <A, B> pura(b: B) = {_:A ->b}
2769. }
2770. fun <A,B,C> ((A) -> B).map(transform:(B) -> C): (A) -> C = {t -> transform(this(t))}
2771. fun <A,B,C> ((A) -> B).flatMap(fm:(B) -> (A) -> C): (A) -> C = {t -> fm(this(t))(t)}
2772. fun <A,B,C> ((A) -> B).ap(fab:(A)->(B) -> C): (A) -> C = fab.flatMap{f -> map(f)}
2773.
2774. fun main()
2775. {
2776. val sumCu5SiMulCu7: (Int) -> Int = {i: Int -> i+5}.ap{{j:Int -> j*7}}
2777. println(sumCu5SiMulCu7(7))
2778. println(sumCu5SiMulCu7(8))
2779. println(sumCu5SiMulCu7(9))
2780. }

```

----- generice_functiientensie_corutina_lambda_list -----

```

2781. import kotlinx.coroutines.*
2782. import java.io.File
2783. import java.util.*
2784.
2785. // suspectez ca functia de extensie era pe domeniul corutinelor
2786. suspend fun CoroutineScope.transform(list: List<Int>): List<Int> =
2787. list.asSequence()
2788. .map{it -> it/3}
2789. .map{it -> it*5}
2790. .toList()
2791.
2792. fun main()= runBlocking<Unit> {
2793. //ain't nobody got time for that

```

```

2794. //val scanner=Scanner(System.`in`)
2795. //println("Filename expected:")
2796. val fileName="numbers.txt"//scanner.nextLine()
2797.
2798. val file=File(fileName)
2799. val lines= file.inputStream().bufferedReader().readText()
2800.
2801. val myList= lines.split(regex = Regex("\\W"))
2802. .asSequence()
2803. .map{it->it.toInt()}
2804. .toList()
2805. val res= async{transform(myList)}
2806. print(res.await())
2807.
2808. }

```

hasmap_funcutor_lambda

```

2809. val procesareLambda={x:Int -> 3*x-1}
2810. fun HashMap<Int, Int>.converttoStr() : HashMap<String, String> =
2811. if(size>0){
2812. val MapNou = HashMap<String, String>(size)
2813. for(element in this){
2814. MapNou.put(element.key.toString(),procesareLambda(element.key).toString())
2815. }
2816. MapNou
2817. }else{
2818. val MapEmpty = HashMap<String, String>()
2819. MapEmpty//nu exista emptyHashMap()
2820. }
2821. fun main(){
2822. val HM= hashMapOf(
2823. 3 to procesareLambda(3),
2824. 5 to procesareLambda(5),
2825. 6 to procesareLambda(6), //functia si functorul va lucra numai cu cheile din HashMap-ul acesta
2826. 7 to procesareLambda(7)
2827. )
2828. println(HM.converttoStr())
2829. }
2830. Alternativ, cred
2831. fun main(args:Array<String>) {
2832. var my_map = hashMapOf<Int, Int>()
2833. for (i in 0..100)
2834. {
2835. my_map[i] = i
2836. }
2837. println(my_map)
2838. my_map.values.map { i-> 3 * i - 1 }
2839. .map(Int::toString)
2840. .forEach ( ::println )
2841.
2842. }

```

memoizarea_calculsir_hasmap

```

2843. import java.util.concurrent.ConcurrentHashMap
2844.
2845. fun <A,R> Function1<A,R>.memoized(): (A)->R{
2846. val map=ConcurrentHashMap<A,R>()
2847. return {a->map.getOrPut(a){this.invoke(a)} }
2848. }
2849. fun f(n:Int):Int{
2850. if (n<1)
2851. return 0
2852. else if(n==1)
2853. return 1
2854. else
2855. return f(n-1)+f(n-2)
2856. }
2857. fun main(){
2858. val memoizedFunction={i:Int -> f(i)}.memoized()//memoizare generalizata
2859. print("n=")
2860. val n= readLine()!!.toInt()
2861. for(i in 1..n) {
2862. print("\nf({i})=")
2863. print(memoizedFunction(i))
2864. }
2865. }
2866. sv108 kot101
2867. -----

```

oop_oameni_beau_danseaza

```
2868. import java.util.*
2869.
2870. class Human(name:String, food:String, drink:String, interest:String)
2871. {
2872.     var name = name;
2873.     var food = food;
2874.     var drink = drink;
2875.     var interest = interest;
2876.     fun display()
2877.     {
2878.         println("name: " + name + " food: " + food + " drink: " + drink + " interest: " + interest )
2879.     }
2880. }
2881. fun Question(text:String): List<String>
2882. {
2883.     val list_of_words = text.split("\\W+").toRegex()
2884.     return list_of_words
2885. }
2886. fun DataMiner(humans:List<Human>, sentence: List<String>):Boolean {
2887.     var human_found = false
2888.     var food_negation_found = false
2889.     var food_found = false
2890.     var drink_negation_found = false
2891.     var drink_found = false
2892.     var interest_negation_found = false
2893.     var interest_found = false
2894.     var b=false
2895.     for (human in humans) {
2896.         human_found = false
2897.         food_negation_found = false
2898.         food_found = false
2899.         var drink_negation_found = false
2900.         var drink_found = false
2901.         interest_negation_found = false
2902.         interest_found = false
2903.
2904.         for (element in sentence) {
2905.             //numele
2906.             if (human.name.equals(element) || human_found) {
2907.                 human_found = true
2908.             }
2909.             //mancare
2910.             if(human_found && element.equals("nu")) {
2911.                 food_negation_found = true
2912.             }
2913.
2914.             if (human.food.equals(element)) {
2915.                 food_found = true
2916.             }
2917.             //bautura
2918.             if(food_found && element.equals("nu")) {
2919.                 drink_negation_found = true
2920.             }
2921.
2922.             if (human.drink.equals(element)) {
2923.                 drink_found = true
2924.             }
2925.             //placeri
2926.             if(drink_found && element.equals("nu")) {
2927.                 interest_negation_found = true
2928.             }
2929.
2930.             if (human.interest.equals(element)) {
2931.                 interest_found = true
2932.             }
2933.
2934.             if(drink_negation_found && food_negation_found && interest_negation_found && human_found && !food_found && !drink_found &&
                !interest_found) {
2935.                 b= true
2936.             }
2937.             else
2938.             {
2939.                 if (human_found && interest_found && drink_found && food_found && !food_negation_found && !drink_negation_found &&
                    !interest_negation_found)
2940.                 {
2941.                     b= true
2942.                 }
2943.             }
2944.             return b
2945.         }
2946.     fun main() {
2947.
2948.         val human1 = Human("Alex", "pizza", "vin", "sefii")
2949.         val human2 = Human("Vasile", "pere", "vodka", "barbati")
2950.         val human3 = Human("Ion", "mere", "bere", "femei")
2951.         var humans: List<Human> = listOf(human1, human2, human3)
```

```

2952. //afisam omuleti
2953. for (element in humans) {
2954. element.display()
2955. }
2956.
2957. val read = Scanner(System.`in`)
2958.
2959. println("Enter String:")
2960. val text = readLine()
2961. if(text == null){
2962. println("error")
2963. } else {
2964. val word = Question(text)
2965. val trust = DataMiner(humans, word)
2966. println(trust)
2967. }
2968. }
2969. //sv58 kot86

```

oop_salacurs_enum

```

2970. enum class Randuri{
2971. RAND1,RAND2,RAND3,RAND4,RAND5,RAND6
2972. }
2973. enum class Geamuri{
2974. GeamStanga,GeamDreapta,GeamFata,GeamSpate
2975. }
2976. class Proiector(val id: Int)
2977. {
2978. fun Open()
2979. {
2980. println("Profesorul a deschis proiectorul ${id}")
2981. }
2982. }
2983. class Elev(val nume: String)
2984. {
2985. fun SePregatesteDeTest(rand:Randuri)
2986. {
2987. println("Elevul " + nume + " se aseaza pe randul " + rand)
2988. }
2989. }
2990. class Profesor(val nume:String)
2991. {
2992. fun DeschideProiectorul(p:Proiector)
2993. {
2994. p.Open()
2995. }
2996.
2997. fun AseazaEleviTest(elevi: MutableList<Elev>)
2998. {
2999. for (i in 0..elevi.size-1)
3000. {
3001. elevi[i].SePregatesteDeTest(Randuri.values()[i])
3002. }
3003. }
3004.
3005. fun InchideGeamul(directie: String)
3006. {
3007. when(directie)
3008. {
3009. "stanga"->{
3010. println("Profesorul a deschis: " + Geamuri.GeamStanga)
3011. }
3012. "dreapta"->{
3013. println("Profesorul a deschis: " + Geamuri.GeamDreapta)
3014. }
3015. "fata"->{
3016. println("Profesorul a deschis: " + Geamuri.GeamFata)
3017. }
3018. "spate"->{
3019. println("Profesorul a deschis: " + Geamuri.GeamSpate)
3020. }
3021. else -> println("Nu exista geam acolo")
3022. }
3023. }
3024. }
3025.
3026. class SalaDeCurs(val profesor:Profesor, val elevi: MutableList<Elev>, val proiector:Proiector)
3027. {
3028. fun ExecuteSomething(actiune: String)
3029. {
3030. when(actiune) {
3031. "proiector" -> {
3032. profesor.DeschideProiectorul(proiector)
3033. }
3034. "geam"->{

```

```

3035. profesor.InchideGeamnul("stanga")
3036. }
3037. "test"->{
3038. profesor.AseazaEleviTest(elevi)
3039. }
3040. else->{
3041. println("Actiunea nu a fost dezvoltata inca")
3042. }
3043. }
3044. }
3045. }
3046.
3047. fun main(args:Array<String>) {
3048.
3049. val elev1 = Elev("Alex")
3050. val elev2 = Elev("Marian")
3051. val elev3 = Elev("George")
3052. val elev4 = Elev("Tudor")
3053. val elev5 = Elev("Hara")
3054. val elev6 = Elev("Ionut")
3055. val elevi = mutableListOf<Elev>(elev1,elev2,elev3,elev4,elev5,elev6)
3056.
3057. val proiector = Proiector(1)
3058. val profesor = Profesor("Mihai")
3059.
3060. val salaDeCurs = SalaDeCurs(profesor,elevi,proiector)
3061.
3062. salaDeCurs.ExecuteSomething("proiector")
3063. salaDeCurs.ExecuteSomething("geam")
3064. salaDeCurs.ExecuteSomething("test")
3065. }
3066. //sv65 kot95

```

oop_salaLab_listeMutable

```

3067. open class Om(val nume:String){
3068. fun ScribeTabla(){
3069. println(nume + " scrie tabla")
3070. }
3071. fun Sterge_Tabla()
3072. {
3073. println(nume + " sterge tabla")
3074. }
3075. }
3076. class Elev( nume:String) : Om(nume)
3077. {
3078. fun Asculita_De_Profesor()
3079. {
3080. println("Elevul " + nume + " asculta")
3081. }
3082. }
3083. class Profesor( nume:String) : Om(nume)
3084. {
3085. fun Preda_La_elev()
3086. {
3087. println("Profesorul " + nume + " predă")
3088. }
3089. fun Porneste_Calculatoare( calculatoare: MutableList<Calculator>)
3090. {
3091. calculatoare.forEach { it.Start() }
3092. }
3093. }
3094. class Calculator(val id: Int)
3095. {
3096. fun Start()
3097. {
3098. println("S-a pornit calculatorul: " + id)
3099. }
3100. }
3101. class Sala(val nume: String, val elevi: MutableList<Elev>, val p:Profesor, val calculatoare: MutableList<Calculator>)
3102. {
3103. fun Actioneaza(om:String, actiune: String)
3104. {
3105. when(om)
3106. {
3107. "elev"-> {
3108. when(actiune)
3109. {
3110. "asculta"-> elevi.forEach { it.Asculita_De_Profesor() }
3111. "sterge" -> elevi.forEach{it.Sterge_Tabla()}
3112. "scrie" -> elevi.forEach{it.Scribe_Tabla()}
3113. }
3114. }
3115. "profesor"-> {
3116. when(actiune)
3117. {
3118. "preda"->p.Preda_La_elev()

```

```

3119. "sterge" -> p.Sterge_Tabla()
3120. "scrie" -> p.Scrie_Tabla()
3121. "start" -> p.Porneste_Calculatoare(calculatoare)
3122. }
3123. }
3124. }
3125. }
3126. }
3127. fun main(args:Array<String>) {
3128. val alex= Elev("Alex")
3129. val marian= Elev("Marian")
3130. val hara= Elev("Hara")
3131. val m= mutableListOf<Elev>()
3132. m.add(alex)
3133. m.add(hara)
3134. m.add(marian)
3135. val c1=Calculator(1)
3136. val c2=Calculator(2)
3137. val c3=Calculator(3)
3138. val calculatoare = mutableListOf<Calculator>()
3139. calculatoare.add(c1)
3140. calculatoare.add(c2)
3141. calculatoare.add(c3)
3142. val p=Profesor("George")
3143. val sala=Sala("Sala2",m,p,calculatoare)
3144. sala.Actioneaza("elev","asculta")
3145. sala.Actioneaza("profesor","sterge")
3146.
3147. }
3148. //sv76 kot106

```

persoana_utilizator_agenda_MutableList

```

3149. interface Person
3150. {
3151. fun getName():String
3152. fun getAge():Int
3153. }
3154.
3155. class PersonImplementation(val nume:String, val ani:Int):Person
3156. {
3157. override fun getAge(): Int {
3158. return ani
3159. }
3160. override fun getName(): String {
3161. return nume
3162. }
3163. }
3164.
3165. class Utilizator(val person:Person):Person by person
3166. {
3167. override fun getAge(): Int {
3168. return person.getAge()
3169. }
3170. override fun getName(): String {
3171. return person.getName()
3172. }
3173. }
3174.
3175. class Agenda(val persoane:MutableList<Utilizator>)
3176. {
3177. fun SearchThrough(name:String): String?
3178. {
3179. persoane.forEach { if(it.getName()==name) { return name } }
3180. return null
3181. }
3182. }
3183.
3184. fun main(args:Array<String>) {
3185. val alex = PersonImplementation("Alex", 20)
3186. val vlad = PersonImplementation("vlad", 21)
3187. val tudor = PersonImplementation("tudor", 22)
3188. val alin = PersonImplementation("Alin", 23)
3189. var users= mutableListOf<Utilizator>(
3190. Utilizator(alex),
3191. Utilizator(vlad),
3192. Utilizator(tudor),
3193. Utilizator(alin)
3194. )
3195. val agenda = Agenda(users)
3196. println(agenda.SearchThrough("Alex"))
3197. println(agenda.SearchThrough("Ionut"))
3198. }
3199. ///////////////

```

ST23,Kot635/restanta2/Stefania

```
interface Person
3200. {
3201. fun getName():String
3202. fun getAge():Int
3203. }
3204.
3205. class Utilizator(val nume:String, val ani:Int):Person
3206. {
3207. override fun getAge(): Int {
3208. return ani
3209. }
3210.
3211. override fun getName(): String {
3212. return nume
3213. }
3214. }
3215.
3216. class Agenda(val persoane:MutableList<Person>)
3217. {
3218. fun SearchThroug(name:String): String?
3219. {
3220. persoane.forEach { if(it.getName()==name) { return name } }
3221. return null
3222. }
3223. }
3224.
3225. fun main(args:Array<String>) {
3226.
3227.
3228. val alex=Utilizator("Alex",20)
3229. val vlad=Utilizator("vlad",21)
3230. val tudor=Utilizator("tudor",22)
3231. val alin=Utilizator("Alin",23)
3232. var persons = mutableListOf<Person>(alex,vlad,tudor,alin)
3233. val agenda = Agenda(persons)
3234. println(agenda.SearchThroug("Alex"))
3235. println(agenda.SearchThroug("Ionut"))
```

sortare_cu_arbore

```
3236. class ArboreBinar<T>(var value:T){
3237. var stanga: ArboreBinar<T>?=null
3238. var dreapta:ArboreBinar<T>?=null
3239. fun <U> map(f:(T) -> U):ArboreBinar<U>{
3240. val arbore=ArboreBinar<U>(f(value))
3241. if(stanga!=null) arbore.stanga=stanga?.map(f)
3242. if(dreapta!=null) arbore.dreapta=dreapta?.map(f)
3243. return arbore
3244. }
3245. fun AfisezParteaSuperioara()="(${stanga?.value},$value,${dreapta?.value})"
3246. fun insert(value: T)
3247. {
3248. if(this==null)
3249. {
3250. this = ArboreBinar(value)
3251. }
3252.
3253. }
3254. }
3255. fun treeSort(list:List<Int>)
3256. {
3257. var tree = ArboreBinar<Int>(list[0])
3258. var SubList = list.subList(1,list.size)
3259. for (elem in SubList)
3260. {
3261.
3262. }
3263.
3264.
3265. }
3266.
3267. fun main()
3268. {
3269. val list= listOf(10,6,20,3,33,8,11)
3270.
3271. }
```

submultimi_cuvinte_txt_lambda(colectii)

```
3272. import java.io.File
3273. val procesare={ str:String-> str.substring(str.length/2-1,(str.length/2)+1)}
3274. val stergere={str:String-> str.substring(2)}
3275. fun main()
3276. {
3277.   val filename="Text.txt"
3278.   val file = File(filename)
3279.   val exists=file.exists()
3280.   if(!exists)
3281.   {
3282.     println("Fisierul nu exista.Se va crea unul nou")
3283.     file.createNewFile()
3284.     file.writeText("Salut Lume!\nPlec din aceasta lume.")
3285.   }
3286.   var cuvinte:MutableList<String> = file.readText().split(" ",".", "!", "\n", ",", "").toList() as MutableList<String>
3287.   cuvinte.removeIf {
3288.     it==" " //eliminam spatii libere
3289.   }
3290.   println(cuvinte)
3291.
3292.   var final=cuvinte.filter { it.length>3 }.map{procesare(it)}//pt kot39
3293.   println(final)
3294.   var final2= cuvinte.filter { it.length>3 }.map{stergere(it)}//pt Kot 40
3295.   println(final2)
3296. }
```

//sv33kot39/ sv32/kot40

submultimiAB_sumcu5siMulcu7

```
3297. //object functie{
3298. // fun <A, B> pura(b: B) = {_:A ->b}
3299. //} //functie ca un aplicative
3300. fun <A,B,C> ((A) -> B).map(transform:(B) -> C): (A) -> C = {t -> transform(this(t))} //functii extensie
3301. fun <A,B,C> ((A) -> B).flatMap(fm:(B) -> (A) -> C): (A) -> C = {t -> fm(this(t))(t)}
3302. fun <A,B,C> ((A) -> B).ap(fab:(A)->(B) -> C): (A) -> C = fab.flatMap{f -> map(f)}
3303.
3304. fun main()
3305. {
3306.   val sumCu5SiMulCu7: (Int) -> Int = {i: Int -> i+5}.ap{{j:Int -> j*7}}
3307.   println(sumCu5SiMulCu7(7))
3308.   println(sumCu5SiMulCu7(8))
3309.   println(sumCu5SiMulCu7(9))
3310. }
3311. //kot31
3312.
```