Examen scris la Algoritmică și Programare
MODEL

Timp de lucru: **1 oră și 30 minute**. Fiecare subiect valorează **0.75 puncte**.

1. Completați spațiile subliniate astfel încât funcția de mai jos să returneze o listă de numere impare.

```
def impare(___):
        return [___ for ___ in range(1, ___, 2)]
```

2. Scrieți o funcție care primește ca parametru o listă de numere întregi și le afișează **pe o singură linie** separate prin virgulă.

3. Rescrieți funcția de mai jos folosind un **for** în loc de **while**:

```
def my_zip(list1, list2):
    result = []
    while list1 != []:
        result.append([list1[0], list2[0]])
        list1, list2 = list1[1:], list2[1:]
```

4. Rescrieți funcția de la **punctul 3** folosind **list comprehensions**:

5. Rescrieți funcția de la **punctul 3** folosind **recursivitate**:

6. Scrieți o funcție **my_zip3** care extinde funcția de la **punctul 3** pentru 3 liste. Puteți folosi orice abordare.

7. Scrieți 3 aserțiuni pentru a testa funcția de la **punctul 6**.

8. Implementați funcția de mai jos conform specificațiilor:

```
def invdiv(nr):
    # returnează True dacă nr (de tip int) este divizibil cu inversul său și False în caz contrar
```

9. Completați specificațiile funcției de mai jos:

```
def cezar(lista, k):
    # Returnează _____
    return map(lambda nr: nr + k, lista)
```

10. Considerăm un program scris obiectual folosind o arhitectură stratificată. Încercuiți toate afirmațiile **false**:
    a. Sortările se vor face pe layer-ul de presentation
    b. Service-urile vor primi ca parametru în constructor minim un repository
    c. Nu vom testa interacțiunea cu utilizatorul folosind assert-uri
    d. Clasele entitate nu pot conține setteri
    e. Repository cu fișiere se poate implementa doar folosind **pickle**

11. Calculați complexitatea ca timp și spațiu suplimentar pentru funcția de mai jos:

```
def f(n):
    if n == 0:
        return 0
    return 1 + 2*f(n-1)
```

12. Implementați o clasă **NumărComplex** care suportă **adunare prin operatorul +** și **afișare user-friendly ca string**

13. Reprezentați printr-o diagramă UML următoarele elemente:
    a. Clasa **A** moștenește clasa **B**
    b. **A** are o metodă **fA** care primește ca parametru un număr întreg și returnează bool
    c. Clasa **C** folosește **B**

14. Asociați algoritmii cu complexitățile. Alegeți tot ce se aplică:

Quicksort	$O(n^2)$
Căutare binară	O(log n)
Bubblesort	O(n log n)

15. Scrieți o funcție **eficientă** care determină dacă o listă dată ca parametru este ordonată crescător.