Lista dotyczy zagadnień wykorzystania kanonicznych funkcji wyższego rzędu służących do przetwarzania list.

**Currying i częściowe aplikowanie funkcji**

Zadanie rozwiąż w wybranym języku: Scala lub Ocaml.

0. Stosując technikę curryingu i częściowej aplikacji funkcji, napisz skonstruuj funkcję **Log** o następujących wymaganiach:

* 1. funkcja przyjmuje na wejściu ciąg znaków *prefix* i zwraca funkcjęktóra:
     1. przyjmuje na wejściu czas i datę *datetime* i zwraca funkcję, która:
        + przyjmuje na wejściu ciąg znaków *text* i wypisuje na wyjście standardowe tekst postaci:
          - [{*prefix*}] *{datetime} \t* {text}

Dla przykładu, program powinien móc realizować poniższy pseudokod:

WarnLog ← Log("Warn")  
NighlyWarnLog ← WarnLog("2022-10-26 01:45")

NightlyWarnLog("Hello")

który wypisze na wyjście standardowe ciąg znaków  
[WARN] 2022-10-26 01:45 Hello

Zastanów się, jak zaimplementować – zgodnie z paradygmatem funkcyjnym, np. z wykorzystaniem curry'ingu – można by zaimplementować możliwość wyświetlanie logów różnego typu (np. DEBUG, WARN, INFO, CRITICAL) w różnych kolorach.

**Własne implementacje mapowania, redukcji i filtrowania**  
**Zadania 1-3 należy wykonać zarówno w języku Scala, jak również OCaml.**

1. Skonstruuj własną implementację funkcji odwzorowującej: **Map.** Funkcja powinna:  
   1. przyjmować na wejściu listę elementów [*e1*, *e2*, … , *en*] oraz
   2. przyjmować na wejściu dowolną, unarną funkcję *f*,
   3. aplikować tę funkcję na każdym elemencie listy,
   4. zwracać na wyjściu nową listę, stanowiącą odwzorowanie listy wejściowej [f(*e1*), f(*e2*), …, f(*en*)]

W rozwiązaniu zalecane jest wykorzystanie rekurencji i dopasowywania do wzorca. Zabronione jest używanie bibliotecznych implementacji funkcji i metod realizujących operację mapowania.

1. Skonstruuj własną implementację funkcji filtrującej: Filter. Funkcja powinna  
   1. przyjmować na wejściu listę elementów [*e*1, *e*2, … , *e*n] oraz
   2. przyjmować na wejściu dowolny predykat, tj. unarną funkcję*pred* zwracającą wartość boolowską,
   3. aplikować funkcję*pred*na kolejnych elementach listy
   4. zwracać na wyjściu nową listę, w której znajdą się tylko te elementy, które spełniają predykat *pred*.

W rozwiązaniu zalecane jest wykorzystanie rekurencji i dopasowywania do wzorca. Zabronione jest używanie bibliotecznych implementacji funkcji i metod realizujących operację filtrowania.

1. Skonstruuj własną implementację funkcji redukującej: **Reduce.** Funkcja powinna:  
   1. przyjmować na wejściu listę elementów [*e*1, *e*2, … , *e*n] oraz
   2. przyjmować na wejściu dowolną, binarną funkcję *op*, funkcja powinna zwracać wartość,
   3. przyjmować na wejściu wartość początkową *acc* (akumulator),
   4. zwracać skalarną wartość, stanowiącą wynik kolejnych aplikacji binarnej funkcji *op* na kolejnych elementach listy i akumulatorze.

Przykładowo, aplikacja funkcji redukującej dla *op* :*=* (+) na liście [1 ; 2 ; 3] powinna być ewaluowana do liczby 6.

Zastanów się, czy kolejność operandów w funkcji *op* ma znaczenie. Zabronione jest używanie bibliotecznych implementacji funkcji i metod realizujących operację redukcji/zwijania/kombinowania (np. fold, fold\_right, etc.).

**Implementacje funkcji z wykorzystaniem kanonicznych funkcji wyższego rzędu**

Implementację każdej z poniższych funkcji należy oprzeć o wykorzystanie albo własnoręcznie napisanych, albo bibliotecznych funkcji implementujących kanoniczne funkcje wyższego rzędu (mapowanie, zwijanie, filtrowanie, etc.). Należy przedstawić rozwiązanie zarówno w języku OCaml, jak również Scala.

1. Napisz funkcję, która przyjmuje na wejściu listę liczb całkowitych, a na wyjściu zwraca średnią (jako liczbę rzeczywistą).
2. Napisz funkcję, która transformuje ciąg znaków zawierający spacje w akronim (np. "Zakład Ubezpieczeń Społecznych" → "ZUS")
3. Napisz funkcję, która przyjmuje na wejściu listę liczb całkowitych, a na wyjściu zwraca kwadraty tych liczb, których sześciany nie są większe od sumy wszystkich liczb.