# PP - kolos

---

Aby zapisac funkcje rekurencyjna potrzebujemy 2 skladowych:

- Przypadek bazowy
- Kork rekurencyjny

Head (glowa) - 1 element listy Tail (ogon) - reszta listy

`W Juli indeksujemy od 1 a nie od 0`

**Najprostszym zobrazowaniem uzytecznosci rekurencji jest obliczenie silni**

```
function silnia(n)
    # Silnia z 0 to 1. To zatrzymuje rekurencję.
    if n == 0
        return 1

    # --- Krok rekurencyjny ---
    # n * (n-1)!
    elseif n >= 0
        return n * silnia(n - 1)
    end
end
```

---

## Dlugosc listy

```
function dlugosc_listy(lista)
    if isempty(lista)
        return 0
    else
        # 1 + długość reszty listy
        return 1 + dlugosc_listy(lista[2:end])
    end
end
```

## Czy lista posiada element

```
function czy_zawiera(lista, liczba)
    # Przypadek bazowy 1: Lista pusta
    if isempty(lista)
        return false
    # Przypadek bazowy 2: Znaleziono liczbę
    elseif lista[1] == liczba
        return true
    # Krok rekurencyjny: Szukaj w reszcie listy
    else
        return czy_zawiera(lista[2:end], liczba)
    end
end
```

## Odwrocenie listy

```
function odwroc(lista)
    if isempty(lista)
        return []
    else
        return [odwroc(lista[2:end])...; lista[1]]
    end
end
```

## Min i Max z listy

```
function findMinMax(lista, currentIdx=1)

    if isempty(lista)
        return (-Inf, Inf)
    elseif currentIdx == length(lista)
        # Ostatni element jest jednocześnie min i max
        el = lista[currentIdx]
        return (el, el)
    end

    # === Krok rekurencyjny ===
    currEl = lista[currentIdx]

    # Wywołaj rekurencję dla tail
    (minVal, maxVal) = findMinMax(lista, currentIdx + 1)

    minVal = (currEl < minVal) ? currEl : minVal
    maxVal = (currEl > maxVal) ? currEl : maxVal

    return (minVal, maxVal)
end

liczby = [0, 15,-33, 100, 53333, -1]
wynik = findMinMax(liczby)
```

## Liczba unikalnych wartosci na liscie

```
function policz_unikalne(lista)
    if isempty(lista)
        return 0
    end

    h = lista[1]
    t = lista[2:end]

    if h in t
        # Jeśli h jest w ogonie, zostanie policzone później
```

```
        return policz_unikalne(t)
    else
        # Jeśli h nie ma w ogonie, liczymy je
        return 1 + policz_unikalne(t)
    end
end
```

## Usuwanie duplikatow z listy

```
function usun_duplikaty(lista)
    if isempty(lista)
        return []
    end

    head = lista[1]
    tail = lista[2:end]

    # 1. Rekurencyjnie usuń duplikaty z ogona
    tailUnique = usun_duplikaty(tail)

    # 2. Sprawdź, czy głowa jest już w oczyszczonym ogonie
    if czy_zawiera(tailUnique, head)
        # Jest duplikatem, pomiń
        return tailUnique
    else
        # Nie jest duplikatem, dołącz ją na początek
        return [head; tailUnique] #
    end
end
```

## Zliczanie wystapienia el na liscie

```
function policz_wystapienia(lista, element)
    if isempty(lista)
        return 0
    end

    head = lista[1]
    tail = lista[2:end]

    if head == element
        return 1 + policz_wystapienia(tail, element)
    else
        return policz_wystapienia(tail, element)
    end
end
```

## Generowanie k liczb losowych z przedzialu (tu [0-100])

```
function generuj_losowe(k)
    # Jeśli k jest równe 0 (lub mniejsze), nie ma już liczb do wygenerowania.
    if k <= 0
        return []
    else
        # 1. Wygeneruj jedną liczbę losową z przedziału [0, 100]
        randNr = rand(0:100)

        # Jezeli podamy k = 1 to wywolanie bedzie typu generuj_losowe(0) co zatrzyma
nam dalsze wywolywanie funkcji zwracajac []
        reszta_listy = generuj_losowe(k - 1)

        # Połącz wygenerowaną liczbę (jako pierwszy element)
        #  z listą pozostałych liczb i zwróć wynik.
        return [randNr; reszta_listy]
    end
end
```

## Lista par: element i jego liczebność (bez powtórzeń)

```
function policz(el, lista)
    if isempty(lista)
        return 0
    end
    return (lista[1] == el ? 1 : 0) + policz(el, @view lista[2:end])
end

function lista_par(lista)
    if isempty(lista)
        return []
    end

    h = lista[1]
    t = @view lista[2:end]

    # Zlicz h w całej liście (ważne: przekazujemy całą 'lista')
    liczba_h = policz(h, lista)

    # Przetwarzaj resztę listy bez elementu h
    reszta_bez_h = filter(x -> x != h, t)
    rekurencyjne_pary = lista_par(reszta_bez_h)

    # Zwróć parę dla h oraz resztę par
    return [(h, liczba_h), rekurencyjne_pary...]
end

# Przykład:
# lista_par([:a, :b, :a, :c, :b, :a])  # Wynik: [(:a, 3), (:b, 2), (:c, 1)]
```

## Różnica dwóch zbiorów (list)

```
function roznica_list(listaA, listaB)
    # roznica zbiorow A i B to zbior elementow nalezacych do A ale nienalezacych do B
    if isempty(listaA)
        return []
    end

    h = listaA[1]
    t = listaA[2:end]

    if h in listaB
        return roznica_list(t, listaB)
    else
        return [h, roznica_list(t, listaB)...]
    end
end

println(roznica_list([1, 2, 3, 4], [2, 4, 6]))
```

## Zamień listę na listę n takich list

```
function powiel_liste(lista, n)
    if n <= 0
        return []
    end

    return [lista, powiel_liste(lista, n-1)...]

end

println(powiel_liste([3,2,1], 5))
```

## Z listy list wybierz listę o największej sumie + suma listy

```
function sumaListy(lista)
    if isempty(lista)
        return 0
    end

    return lista[1] + sumaListy(lista[2:end])
end

function maxSuma(lista)
    if isempty(lista)
        return -Inf
    end

    if length(lista) == 1
        return lista[1]
    end
```

```
    h = lista[1]
    t = lista[2:end]

    najlepszaReszta = maxSuma(t)

    suma_h = sumaListy(h)
    suma_najlepszejReszty = sumaListy(najlepszaReszta)

    return suma_h > suma_najlepszejReszty ? h : najlepszaReszta
end

println(maxSuma([[1, 2], [10, -5], [3, 3, 3]]))
```

## Przenieś elementy mniejsze niż k na początek

```
function mniejsze_od_k_na_poczatek(lista, k)
    if isempty(lista)
        return []
    end

    h = lista[1]
    t = lista[2:end]

    reszta = mniejsze_od_k_na_poczatek(t, k)

    if h < k
        return [h, reszta...]
    else
        return [reszta..., h]
    end

end

println(mniejsze_od_k_na_poczatek([5, 1, 8, 2, 6, 3], 5))
```

## Na liście list znajdź listę zawierającą podany element

```
# Zwraca pierwszą pod-listę, która zawiera szukany element.
function znajdzWListach(lista, el)
    if isempty(lista)
        return []
    end

    if length(lista) == 1
        return lista[1]
    end

    h = lista[1]
    t = lista[2:end]
```

```
    if el in h
        return h
    else
        return znajdzWListach(t, el)
    end
end

println(znajdzWListach([[1, 2], [3, 4, 1], [5, 6, 7, 4]], 4))
```

## Zamień liczbę na napis (np. "5 ... 4 ... 3 ... 2 ... 1 ... !")

```
# Przykład:
# liczba_na_napis(5)  # Wynik: "5 ... 4 ... 3 ... 2 ... 1 ... !"

function zmienNaNapis(nr)
    if nr <= 0
        return "!"
    elseif nr == 1
        return "1...!"
    else
        return "$(nr)...$(zmienNaNapis(nr-1))"
    end
end
```

## Wskazany element przenieś na początek listy

```
function przenies_na_poczatek(el, lista)
    if isempty(lista)
        return []
    end

    h = lista[1]
    t = lista[2:end]

    if h == el
        return lista
    end


    # h !== el wiec szukamy w ogonie
    wynik_ogona = przenies_na_poczatek(el, t)


    ## Jeśli 'wynik_ogona' zaczyna się od 'el', to znaczy, że
    # 'el' został znaleziony gdzieś głębiej i jest teraz na czele ogona.
    if !isempty(wynik_ogona) && wynik_ogona[1] == el
        reszta_ogona = wynik_ogona[2:end]
```

```
        return [el, h, reszta_ogona...]
    else
        # 'el' nie został znaleziony w ogonie.
        return [h, wynik_ogona...]
    end
end
```

## Oblicz średnią wartość elementów listy

```
function _sredniaHelper(lista, currSum, counter)
    if isempty(lista)
        if counter == 0
            return 0
        else
            return currSum / counter
        end
    end

    h = lista[1]
    t = lista[2:end]

    return _sredniaHelper(t, currSum+h, counter+1)
end

function srednia(lista)
   return _sredniaHelper(lista, 0, 0)
end

println(srednia([2,5,3]))
```

## Sprawdź, czy dwie listy mają przynajmniej jeden wspólny element

```
function czyWspolny(listaA, listaB)
    if isempty(listaA)
        return false
    end

    h = listaA[1]
    t = listaA[2:end]

    if h in listaB
        return true
    else
        return czyWspolny(t, listaB)
    end
end
```

## Sprawdź, czy dwie listy są rozłączne

```
# Dwie listy są rozłączne, jeśli nie mają ani jednego wspólnego elementu.

function czyRozlaczne(listaA, listaB)

    if isempty(listaA)
        return true # Pusta lista jest rozłączna ze wszystkim
    end

    h = listaA[1]
    t = listaA[2:end]

    if h in listaB
        return false
    else
        return czyRozlaczne(t, listaB)
    end
end

println(czyRozlaczne([2,3,5], [1,8,7]))
```

## Listę list zamień na listę sum

```
function sumaListy(lista)
    if isempty(lista)
        return 0
    end

    return lista[1] + sumaListy(lista[2:end])
end

function zmienListeListyNaSumt(lista)
    if isempty(lista)
        return []
    end

    h = lista[1]
    t = lista[2:end]

    suma_h = sumaListy(h)

    return [suma_h, zmienListeListyNaSumt(t)...]
end
```