Употреба рекурзија

Рекурзија никада није неопходна да би се решио некакав програмерски проблем.
Други начин решавања проблема је коришћењем програмерских петљи.
Иначе, рекурзивни алгоритми се мање користе од итеративних алгоритама пошто процес позивања функције захтева више радњи од стране рачунара (алокација меморије и смештање адреса локација у меморију где се контрола враћа после извршења функције).
Ове акције се називају **прекоброjне** (overhead), одвијају се увек када се функција позове и оне се не реализују у случају петљи.
Али, неки проблеми се лакше решавају коришћењем рекурзија него петљи; генерално тамо где се петља брзо извршава, могуће је направити брз рекурзиван алгоритам.
Принцип употребе рекурзија: ако је проблем решив одмах без рекурзије, функција га решава; ако проблем не може бити решен одмах, онда га функција смањује на мање али сличне проблеме, па позива саму себе да би решила те мање проблеме.
Прво се идентификује најмање један случај у којем се проблем може решити без рекурзије, и то се назива **основни случај** (base case).
Друго, одреди се начин решавања проблема за све друге случајеве коришћењем рекурзије и то се назива **рекурзивни случај** (recursive case).
У овом кораку, увек се мора смањити проблем на мању верзију оригиналног проблема.
Смањивањем проблема са сваким рекурзивним позивом, основни случај ће бити у одређеном моменту достигнут и рекурзија ће се зауставити.

Рачунање факторијала коришћењем рекурзије

У математици, писање n! означава факторијел броја n.
Факторијел ненегативног броја се може дефинисати:
Ако је n = 0 онда је n! = 1; ако је n > 0 онда је n! = 1 x 2 x 3 x ... x n.
За употребу рекурзије довољно је уместо n! користити функцију: faktorijel(n)
При дизајну рекурзивног алгоритма који рачуна факторијел било којег броја, прво се идентификује основни случај, који је део израчунавања који се може решити без употребе рекурзије (ако је n = 0 онда је faktorijel(n) = 1).
Овиме се решава проблем када је n једнако 0, али како се решава проблем за n веће од 0?
То је рекурзиван случај, или део проблема који се решава коришћењем рекурзије.
Ако је n > 0 онда је faktorijel(n) = n x faktorijel(n - 1)
Сада позив рекурзији ради са смањеном верзијом проблема, n – 1.
Зато се рекурзивно правило може написати и овако:
Ако је n = 0 онда је faktorijel(n) = 1; ако је n > 0 онда је faktorijel(n) = n x faktorijel(n - 1).

**0252 Рачунање факторијала рекурзијом**

```
def main():
    broj = int(input('Uneti nenegativan broj: '))
    x = faktorijal(broj)
    print('Faktorijal od', broj, 'je', x)

def faktorijal(n):
    if n == 0:
```

```
        return 1
    else:
        return n * faktorijal(n - 1)

main()
```

Команда return се неће одмах извршити, пошто се прво мора реализовати позив функције faktorijal(n - 1).
Функција faktorijal саму себе позива пет пута, све док се параметар n не постави на 0.
Рекурзивни алгоритам мора смањити проблем са сваким рекурзивним позивом, а да би се добило решење рекурзија се мора у једном моменту зауставити и то на основном случају (пошто он не захтева рекурзију за своје решавање).

**0253 Сумирање рекурзијом**

```
def main():
    brojevi = [1, 2, 3, 4, 5, 6, 7, 8, 9]
    moja_suma = opseg(brojevi, 2, 5)
    print('Suma izmedju pozicija 2 i pozicije 5 je', moja_suma)

def opseg(broj_lista, start, kraj):
    if start > kraj:
        return 0
    else:
        return broj_lista[start] + opseg(broj_lista, start + 1, kraj)

main()
```

У примеру је функција opseg() која користи рекурзију да би се сабрао опсег бројева унутар листе.
Функција користи аргументе: broj_lista листу која садржи опсег вредности за сабирање, start целобројну вредност која је индекс почетног елемента у опсегу, kraj целобројну вредност која указује на индекс крајњег елемента у опсегу.
Основни случај функције је када параметар start је већи од kraj параметра.
Ако је то тачно, функција враћа вредност 0, иначе функција враћа исказ после наредбе return.
Тај исказ враћа збир `broj_lista[start]` и повратну вредност рекурзивног позива.
Приметити да је у позиву рекурзије, почетни елемент у опсегу start + 1.
Заправо, овај исказ враћа вредност првог елемента у опсегу сабран са збиром преосталих елемената у опсегу.