Увод у рекурзије

Рекурзивне функције су функције које позивају саме себе.

Пример 01:

```
def main():
    poruka()

def poruka():
    print('Ovo je rekurzivna funkcija.')
    poruka()

main()
```

Као резултат добија се понављање поруке све док Пајтон не прекине њено понављање.

Функција poruka() позива саму себе и циклус извршавања функције се понавља.

Први проблем са овим начином креирања функција је што се рекурзивна функција не може зауставити и на овај начин личи на бесконачну петљу.

Да би се могла успешно користити, рекурзивна функција мора имати начин контролисања броја пута понављања.

Следећи пример је модификована верзија претходног примера, само што сада функција poruka() добија аргумент који одређује колико пута ће се порука исписати на екрану.

Пример 02:

```
def main():
    poruka(5)

def poruka(broj_ponavljanja):
    if broj_ponavljanja > 0:
        print('Ovo je rekurzivna funkcija.')
        poruka(broj_ponavljanja - 1)

main()
```

Функција poruka() у себи садржи if исказ који контролише понављање поруке.

Све док је параметар broj_ponavljanja већи од 0, порука ће се приказати, функција позива саму себе али се вредност аргумента смањује.

Сваки пут када се функција poruka() позове, нова инстанца параметра broj_ponavljanja се креира у меморији и добија вредност која је за један мања од претходне.

То значи да ће де одиграти 6 позива функције poruka() и да ће у последњем позиву параметар broj_ponavljanja бити једнак 0 што значи да више услов за позивање функције није испуњен.

Функција poruka() се само једном позива из main() функције а остали позиви су када функција позива саму себе.

Број позива функције када позива саму себе се назива **дубина рекурзије**.

У овом примеру, дубина рекрзије је 5.

Када се изведе 6.позив функције poruka(), извршава се прва линија кода испод линије позива, а у овом случају нема линије кода.

То значи да се после извршавања 6.позива, контрола функције враћа на 5.позив функције.

Овакво понављање се извршава уназад до 1.позива функције.

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 свлучају 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.