Rekurze

co je to rekurze?

- víme, že v pythonu může funkce volat jinou funkci
- volá-li funkce sama sebe, nazývá se rekurzivní funkcí

```
def rekurze():
    if podminka:
    else:
        rekurze()
```

- rekurzivní funkce potřebuje podmínku pro své ukončení, jinak pojede v nekonečném cyklu
- nejčastěji se ukončení funkce definuje pomocí if podmínky

co je to rekurze? - počet iterací

- nejčastěji má python nastavený limit pro počet rekurzívních operací nastavený na 1000
- v jupyteru to bývá nastaveno na 3000

```
from sys import getrecursionlimit
getrecursionlimit()
```

```
3000
```

```
from sys import setrecursionlimit
setrecursionlimit(1000)
getrecursionlimit()
```

```
1000
```

rekurze v praxi

Napišme si funkci, která odpočítá od daného čísla *n* do 0.

```
def odpocet(n):
    print(n)
    odpocet(n-1)

odpocet(5)
```

- takováto funkce bude odpočítávat od 5 až do svého limitu **RecursionError:**

```
def odpocet(n):
    print(n)
    if n > 0:
        odpocet(n-1)

odpocet(5)
```

rekurze v praxi (2)

- je dobrou praxí používat ***if*** větev na případ bez rekurze a ***else*** větev funkce pro rekurzi

```
def odpocet(n):
    if n == 0:
        print(n)
    else:
        print(n)
        odpocet(n-1)

odpocet(6)
```

Cvičení 1.

Napište rekurzivní funkci s if podmínkou, která vypíše všechny sudé čísla od *n* do 0, kde *n* je argument funkce.

nápověda: modulo operátor se v pythonu značí **%**

- modulo ukáže zbytek po celočíselném dělení

```
7 % 2
```

```
1
```

```
6 % 2
```

```
0
```

rekurze v praxi (3)

- s každým rekurzivním voláním funkce vznikají nový prostor s lokálními proměnnými a na které vidí jen ona sama, čili:

```
def suma(n):
    if n > 0:
        return n + suma(n-1)
    return 0

print(suma(3))
```

```
suma(3)                 #první zavolání funkce
3 + suma(2)             #druhé zavolání funkce
3 + 2 + suma(1)         #třetí zavolání funkce
3 + 2 + 1 + suma(0)     #čtvrté zavolání funkce
0                       #vrácení hodnoty ze čtvrtého zavolání funkce (suma(0) vrátí 0)
1 + 0                   #vrácení hodnoty z třetího zavolání funkce (suma(1) vrátí 1)
2 + 1                   #vrácení hodnoty z druhého zavolání funkce (suma(2) vrátí 3)
3 + 3                   #vrácení hodnoty z prvního zavolání funkce (suma (3) vrátí 6)
```

Cvičení 2.

Napište rekurzivní funkci s ***if*** podmínkou, která počítá faktoriál čísla *n*,
kde *n* je argument funkce.

nápověda: faktoriál 3! = 3*2*1 = 6

svoji funkci provnejte s:

```
from math import factorial
factorial(n)
```

Cvičení 3.

Napište rekurzivní funkci s if podmínkou, která vypíše *n*-té číslo Fibonacciho posloupnosti, kde *n* je argumentem funkce.

nápověda:

```
fibonacci(0) = 0
fibonacci(1) = 1
fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)
```

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597

rychlost rekurzivní funkce = bývají pomalé

pro zjištění rychlosti funkce můžeme použít:

```
setup_string = "from math import factorial"

from timeit import timeit
timeit("factorial(4)", setup=setup_string, number=10000000)
```

```
0.2454161000005115
```

```
setup_string = """def factorial2(n):
    if n == 1: # break-down condition
        return 1
    else:
        return n * factorial2(n-1) """

from timeit import timeit
timeit("factorial2(4)", setup=setup_string, number=10000000)
```

```
2.188432599999942
```

dva type rekurze - 1. Přímá (ang. Direct)

- typ rekuze, který volá sám sebe (s tímhle typem jsem pracovali doteď)

```
def odpocet(n):
    if n == 0:
        print(n)
    else:
        print(n)
        odpocet(n-1)

odpocet(6)
```

dva type rekurze - 1. Nepřímá (ang. Indirect)

- typ rekuze, který volá jinou funkci, která volá funkci původní

```
def A(n):
    if n > 0:
        print("", n, end="")
        B(n + 1)

def B(n):
    if n > 1:
        print("", n, end="")
        A(n - 5)

A(20)
```

```
 20 21 16 17 12 13 8 9 4 5
```

Cvičení 4.

Napište rekurzivní funkci s if podmínkou, která vypíše největšího společného dělitele (NSD) dvou čísel.

nápověda:

a. použijte operátor modulo %
b. použijte Eukleidův algoritmus
    i. pokud je a=0 pak NSD(0,b)=b, pokud je b=0 pak NSD(a,0)=a
    ii. spočítejte zbytek po celočíselném dělení = c
    iii. využijte platnosti NSD(a,b)=NSD(b,c)

Cvičení 5.

Napište rekurzivní funkci se dvěma argumenty, která ověří jestli první z argumentů je mocninou druhého argumentu.

nápověda (1):

```
def je_mocninou(a, b):
    ...

print(je_mocninou(64, 2)) #True
print(je_mocninou(1000, 10)) #True
print(je_mocninou(1001, 10)) #False
```

nápověda (2):

použijte operátor modulo %

použijte podmínku ***if*** dvakrát

Cvičení 6.

Napište rekurzivní funkci se dvěma argumenty, která spočítá nejmenší společný násobek (NSN).

nápověda:

- NSN dvou čísel je nejmenší kladné celé číslo, které lze vydělit každým z nich
- NSN(12,15) = 60

Napište funkci, která spočítá NSN s využitím funkce pro největší společný dělitel NSD.

nápověda: $$NSN\,(a,b) = \frac{a \times b}{NSD\,(a,b)}$$

Cvičení 7.

Napište rekurzivní funkci s jedním argumentem, která vypíše posloupnost Collatzeho problému.

Definice:

- pokud je číslo *n* liché, pak 3n + 1
- pokud je číslo *n* sudé, pak n/2

$$C(n) = \begin{cases} 3n+1, & \text{je-li } n \text{ liché}, \\ n/2, & \text{je-li } n \text{ sudé}. \end{cases}$$

- každá takováto řada konverguje k číslu 1 (pozor z jedničky se stane 4 a ta se zase dostane na 1)