Oznam 1

Vo štvrtok 17.10. je študijné voľno. V stredu 16.10. sa učí podľa štvrtkového rozvrhu, teda cvičenia majú tí študenti, ktorí mali mať cvičenia vo štvrtok.

Tí študenti, ktorí mali mať cvičenia v stredu, majú tento týždeň cvičenia **ospravedlnené** a nemusia sa zúčastniť cvičení.

Ak chcú, môžu prísť na **ktorékoľvek** cvičenie, ktoré v stredu bude, t.j. v časoch od 10:00, 13:00, 15:00, 17:00 v C117.
(odporúčam však, ak je to možné, aby ste si priniesli vlastné počítače).

Oznam 2

Prvý test bude na cvičeniach v týždni 7, teda na cvičeniach:

- streda 30.10. 15:00, 17:00

- štvrtok 31.10. 10:00, 13:00, 15:00, 17:00

- náplňou testu bude všetko, čo preberieme na prednáškach a cvičeniach počas prvých 6 týždňov semestra

PROG1: Prednáška 5
Rekurzia

Volania funkcií

Pre každú funkciu v jazyku Python je nutné rozlišovať:

1) Definícia funkcie – teda jej hlavička (identifikátor funkcie, vstupné parametre) a jej telo – t.j. zdrojový kód funkcie. Definícia funkcie hovorí jazyku Python, **čo funkcia robí**, avšak ešte nedochádza k jej spusteniu.

2) Volanie funkcie – spustenie funkcie s príslušnými vstupnými argumentami – teda konkrétne spustenie funkcie s nejakými vstupmi.

Počas volania funkcie daná funkcia beží, v pamäti sú vytvorené jej premenné s nejakými hodnotami. V momente, keď funkcia skončí, tak sa z pamäte jej lokálne premenné odstránia.

Funkcia môže skončiť alebo vrátením návratovej hodnoty pomocou príkazu **return** alebo tým, že sa vykoná posledný príkaz jej tela.

Volania funkcií

Stretli sme sa viacerými situáciami, že nejaká funkcia volá inú funkciu.

Napríklad na cvičení pre 4. týždeň bola nasledovná úloha:

a) definovať funkciu *delitelnost(a,d)* ktorá pre vstupné parametre *a,d* vráti True, ak číslo *d* delí číslo *a*
b) definovať funkciu *je_prvocislo(n),* ktorá vráti True, ak *n* je prvočíslo, inak vráti False; funkcia má zistiť, či je *n* prvočíslo tak, že otestuje, či je *n* deliteľné nejakým číslom z rozsahu {2, 3, …, *n*-1}. Ak áno, *n* nie je prvočíslo. Ak žiadne číslo z rozsahu {2,3,…,*n*-1} nedelí číslo *n*, potom *n* je prvočíslo. Pre tieto účely sa bude vo funkcii *je_prvocislo(n)* volať funkcia *delitelnost(a,d)*.

Riešenie pozri nasledovný slajd.

Volania funkcií

Funkciu *je_prvocislo(n)* naprogramujeme tak, že vytvoríme pomocnú lokálnu premennú *existoval_delitel*, do ktorej uložíme hodnotu *True* v prípade, že nájdeme nejaké číslo z rozsahu {2,3,…,*n*-1}, ktoré delí číslo *n*.

Keďže na začiatku sme ešte neskontrolovali žiadneho potenciálneho deliteľa, nastavíme premennú *existoval_delitel* na hodnotu *False.*

Následne v cykle postupne prechádzame hodnoty z množiny {2,3,…,*n*-1} (pomocou riadiacej premennej cyklu *d*) a pre každú hodnotu *d* zavoláme funkciu *delitelnost(n,d)*. Ak funkcia *delitelnost(n,d)* vráti hodnotu True, tak potom *d* delí *n* a my nastavíme premennú *existoval_delitel* na hodnotu *True.*

Všimnite si teda, ako v definícii funkcie *je_prvocislo(n)* využívame (voláme) inú funkciu – funkciu *delitelnost(n,d)*

```python
def delitelnost(a,d):
    """
    Funkcia vrati True ak cislo d deli cislo a. Inak vrati False.

    a: cele cislo
    d: cele cislo
    """
    if (a%d == 0):
        return True
    else:
        return False

def je_prvocislo(n):
    """
    Funkcia vrati True ak je vstupny parameter n prvocislo. Inak vrati False.

    n: cele cislo
    """
    existoval_delitel = False
    for d in range(2,n):            #otestujeme, ci n deli nejake cislo d z mnoziny {2,...,n-1}
        if delitelnost(n,d) == True: #ak existuje delitel d, cislo n nie je prvocislo
            existoval_delitel = True
    if existoval_delitel == True:
        return False                  #ak sa nasiel delitel, funkcia vrati False, teda n nie je prvocislo
    else:
        return True                   #ak sa nenasiel delitel, funkcia vrati True, n je prvocislo
```

Ako vyzerá volanie funkcie?

Predstavme si, že sa zavolá funkcia *je_prvocislo()* so vstupným argumentom, číslom 4:

```
def delitelnost(a,d):
    if (a%d == 0):
        return True
    else:
        return False

def je_prvocislo(n):
    existoval_delitel = False
    for d in range(2,n):
        if delitelnost(n,d) == True:
            existoval_delitel = True
    if existoval_delitel == True:
        return False
    else:
        return True

vysledok_testu = je_prvocislo(4)
```

n = 4

Volanie funkcie je_prvocislo() so vstupným argumentom, číslom 4

Zavolá sa funkcia *je_prvocislo(4)* , teda vstupný argument má hodnotu 4.

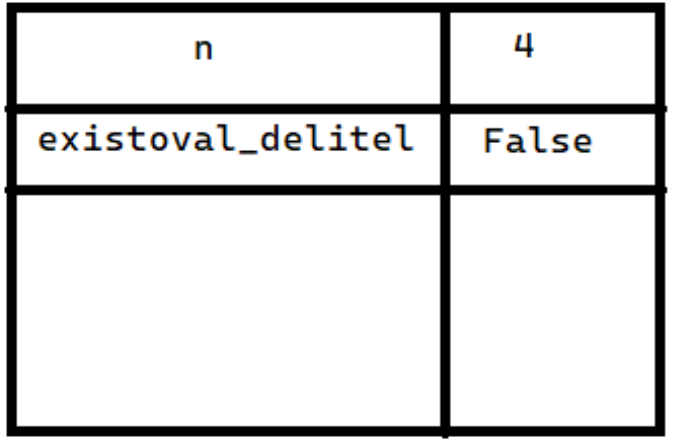
Ako vyzerá volanie funkcie?

Prvý príkaz vo volaní *je_prvocislo(n)* vytvorí premennú *existoval_delitel* s hodnotou False

```
def delitelnost(a,d):
    if (a%d == 0):
        return True
    else:
        return False

def je_prvocislo(n):
    existoval_delitel = False
    for d in range(2,n):
        if delitelnost(n,d) == True:
            existoval_delitel = True
    if existoval_delitel == True:
        return False
    else:
        return True

vysledok_testu = je_prvocislo(4)
```



Ako vyzerá volanie funkcie?

Vytvorí sa riadiaca premenná cyklu, *d* a nadobudne prvú hodnotu z *range(2,4)*, teda 2:

```
def delitelnost(a,d):
    if (a%d == 0):
        return True
    else:
        return False

def je_prvocislo(n):
    existoval_delitel = False
    for d in range(2,n):
        if delitelnost(n,d) == True:
            existoval_delitel = True
    if existoval_delitel == True:
        return False
    else:
        return True

vysledok_testu = je_prvocislo(4)
```

n	4
existoval_delitel	False
d	2

Ako vyzerá volanie funkcie?

Zavolá sa funkcia *delitelnost(a,d)* s argumentami (4,2). Funkcia *je_prvocislo* teda aktuálne **čaká, kým funkcia** *delitelnost(4,2)* vráti návratovú hodnotu!!!

```
def delitelnost(a,d):
    if (a%d == 0):
        return True
    else:
        return False
                a=4        d=2
def je_prvocislo(n):
    existoval_delitel = False
    for d in range(2,n):
        if delitelnost(n,d) == True:
            existoval_delitel = True
    if existoval_delitel == True:
        return False
    else:
        return True

vysledok_testu = je_prvocislo(4)
```

a	4
d	2

n	4
existoval_delitel	False
d	2

Ako vyzerá volanie funkcie?

Na základe aktuálnej hodnoty parametrov, a=4,d=2 funkcia *delitelnost(4,2)* vráti hodnotu *True* a **skončí**.

```
def delitelnost(a,d):
    if (a%d == 0):
        return True
    else:
        return False

def je_prvocislo(n):
    existoval delitel = False
    for d in range(2,n):
        if delitelnost(n,d) == True:
            existoval_delitel = True
    if existoval_delitel == True:
        return False
    else:
        return True

vysledok_testu = je_prvocislo(4)
```

a	4
d	2

n	4
existoval_delitel	False
d	2

Ako vyzerá volanie funkcie?

Po volaní funkcie *delitelnost(4,2)* sa vyhodnotí podmienka, v ktorej sa funkcia volala, t.j. vyhodnotí sa či *delitelnost(4,2)* == *True*. Keďže áno, pokračuje sa v „if“-vetve podmienky.

```
def delitelnost(a,d):
    if (a%d == 0):
        return True
    else:
        return False

def je_prvocislo(n):
    existoval_delitel = False
    for d in range(2,n):
        if delitelnost(n,d) == True:
            existoval_delitel = True
    if existoval_delitel == True:
        return False
    else:
        return True

vysledok_testu = je_prvocislo(4)
```

True

n	4
existoval_delitel	False
d	2

Ako vyzerá volanie funkcie?

V „if“-vetve podmienky sa nastaví premenná *existoval_delitel* na *True* a pokračuje sa s ďalšou iteráciou for-cyklu.

```
def delitelnost(a,d):
    if (a%d == 0):
        return True
    else:
        return False

def je_prvocislo(n):
    existoval_delitel = False
    for d in range(2,n):
        if delitelnost(n,d) == True:
            existoval_delitel = True
    if existoval_delitel == True:
        return False
    else:
        return True

vysledok_testu = je_prvocislo(4)
```

n	4
existoval_delitel	True
d	2

Ako vyzerá volanie funkcie?

Znovu sa vykoná telo cyklu, tentokrát má riadiaca premenná cyklu hodnotu *d* = 3:

```
def delitelnost(a,d):
    if (a%d == 0):
        return True
    else:
        return False

def je_prvocislo(n):
    existoval_delitel = False
    for d in range(2,n):
        if delitelnost(n,d) == True:
            existoval_delitel = True
    if existoval_delitel == True:
        return False
    else:
        return True

vysledok_testu = je_prvocislo(4)
```

n	4
existoval_delitel	True
d	3

Ako vyzerá volanie funkcie?

```
def delitelnost(a,d):
    if (a%d == 0):
        return True
    else:
        return False

def je_prvocislo(n):
    existoval_delitel = False
    for d in range(2,n):
        if delitelnost(n,d) == True:
            existoval_delitel = True
    if existoval_delitel == True:
        return False
    else:
        return True

vysledok_testu = je_prvocislo(4)
```

a=4

d=3

a	4
d	3

n	4
existoval_delitel	True
d	3

Ako vyzerá volanie funkcie?

Na základe aktuálnej hodnoty parametrov, a=4,d=3 funkcia *delitelnost(4,3)* vráti hodnotu *False* a **skončí**.

```
def delitelnost(a,d):
    if (a%d == 0):
        return True
    else:
        return False

def je_prvocislo(n):
    existoval_delitel = False
    for d in range(2,n):
        if delitelnost(n,d) == True:
            existoval_delitel = True
    if existoval_delitel == True:
        return False
    else:
        return True

vysledok_testu = je_prvocislo(4)
```

a	4
d	3

n	4
existoval_delitel	True
d	3

Ako vyzerá volanie funkcie?

Po volaní funkcie *delitelnost(4,3)* sa vyhodnotí podmienka, v ktorej sa funkcia volala, t.j. vyhodnotí sa či *delitelnost(4,3) == True*. Keďže nie, telo „if“-podmienky sa nevykoná a cyklus skončí, pretože sa vykonal posledný príkaz v cykle pre poslednú hodnotu *d*.

```
def delitelnost(a,d):
    if (a%d == 0):
        return True
    else:
        return False

def je_prvocislo(n):
    existoval_delitel = False
    for d in range(2,n):
        if delitelnost(n,d) == True:
            existoval_delitel = True
    if existoval_delitel == True:
        return False
    else:
        return True

vysledok_testu = je_prvocislo(4)
```

n	4
existoval_delitel	True
d	3

Ako vyzerá volanie funkcie?

Vyhodnotí sa podmienka *existoval_delitel == True*. Keďže vo volaní funkcie *je_prvocislo(4)* nastala situácia, že premenná *existoval_delitel* má hodnotu *True*, podmienka je pravdivá. Funkcia teda **vráti** hodnotu *False.*

```
def delitelnost(a,d):
    if (a%d == 0):
        return True
    else:
        return False

def je_prvocislo(n):
    existoval_delitel = False
    for d in range(2,n):
        if delitelnost(n,d) == True:
            existoval_delitel = True
    if existoval_delitel == True:
        return False
    else:
        return True

vysledok_testu = je_prvocislo(4)
```

n	4
existoval_delitel	True
d	3

Ako vyzerá volanie funkcie?

Vyhodnotí sa podmienka *existoval_delitel == True*. Keďže vo volaní funkcie *je_prvocislo(4)* nastala situácia, že premenná *existoval_delitel* má hodnotu *True*, podmienka je pravdivá. Funkcia teda **vráti** hodnotu *False.*

```
def delitelnost(a,d):
    if (a%d == 0):
        return True
    else:
        return False

def je_prvocislo(n):
    existoval_delitel = False
    for d in range(2,n):
        if delitelnost(n,d) == True:
            existoval_delitel = True
    if existoval_delitel == True:
        return False
    else:
        return True

vysledok_testu = je_prvocislo(4)
```

Ako vyzerá volanie funkcie?

Videli sme teda príklad, ako to vyzerá, keď funkcie volajú iné funkcie.

Ak teda nejaká funkcia volá *inú funkciu*, **volajúca funkcia čaká na návratovú hodnotu / ukončenie činnosti** volanej funkcie.

Zároveň platí, že **každé volanie funkcie** má vlastné hodnoty parametrov a lokálnych premenných!

Rekurzia - úvod

Videli sme teda príklad, ako to vyzerá, keď funkcie volajú iné funkcie.

Avšak, v programovaní sa používa aj jav, v ktorom **funkcia volá samú seba!**

Takýto jav nazývame **rekurzia.**

Rekurzia je veľmi silný programátorský nástroj, ktorý umožňuje riešiť mnohé programátorské problémy oveľa elegantnejšie, než pomocou cyklov a zložitejších programátorských konštrukcií.

Avšak ide o pomerne náročnú problematiku, ktorej pochopenie vyžaduje dobrú znalosť toho, ako fungujú funkcie:
- čo to znamená, že funkcia **vracia hodnotu**
- ako fungujú parametre funkcií

Rekurzia - úvod

Čo to vôbec znamená, že „funkcia volá samú seba?“

Zjednodušene povedané, v definícii funkcie – v jej tele - dôjde k volaniu tej istej funkcie, t.j. k volaniu práve tej funkcie, ktorú definujeme.

Čo urobí funkcia *odpocet(n)*, ak ju zavoláme s argumentom 3, t.j. *odpocet(3)* ???

```
def odpocet(n):
    if n <= 0:
        print("Start!")
    else:
        print(n)
        odpocet(n-1)
```

Rekurzia - úvod

```
def odpocet(n):
    if n <= 0:
        print("Start!")
    else:
        print(n)
        odpocet(n-1)
```

```
odpocet(3)
3
2
1
Start!
```

volanie odpocet(3)
1) vypíše „3“
2) zavolá odpocet(2)
    volanie odpocet(2)
    1) vypíše „2“
    2) zavolá odpocet(1)
        volanie odpocet(1)
        1) vypíše „1“
        2) zavolá odpocet(0)
            volanie odpocet(0)
            1) vypíše „Start!“
            2) volanie odpocet(0) skončí
        3) volanie odpocet(1) skončí
    3) volanie odpocet(2) skončí
3) volanie odpocet(3) skončí

Rekurzia - úvod

Vložme do funkcie *odpocet(n)* pomocné výpisy.

1. Prvý výpis sa vykoná hneď po spustení funkcie s príslušným parametrom *n*. Výpis vypíše správu „Začalo sa vykonávanie funkcie *odpocet* s parametrom n = “ a uvedie hodnotu vstupného parametra

2. Druhý výpis sa vykoná ako posledný príkaz vo funkcii. Výpis vypíše správu „Ukončí sa vykonávanie funkcie *odpocet* s parametrom n = “ a uvedie hodnotu vstupného parametra

Všimnite si teda, že **jednotlivé volania sa líšili vstupným parametrom – každé volanie funkcie** teda existuje samostatne s vlastnou hodnotou parametra!!!

Rekurzia - úvod

prednaska5.py - Z:/Documents/prednaska/prednaska5.py (3.12.6)

File Edit Format Run Options Window Help

```
def odpocet(n):
    print("Zacalo sa vykonavanie funkcie odpocet s parametrom n = ",n)
    if n <= 0:
        print("Start!")
    else:
        print(n)
        odpocet(n-1)
    print("Ukonci sa vykonavanie funkcie odpocet s parametrom n = ",n)

odpocet(3)
```

```
Zacalo sa vykonavanie funkcie odpocet s parametrom n =  3
3
Zacalo sa vykonavanie funkcie odpocet s parametrom n =  2
2
Zacalo sa vykonavanie funkcie odpocet s parametrom n =  1
1
Zacalo sa vykonavanie funkcie odpocet s parametrom n =  0
Start!
Ukonci sa vykonavanie funkcie odpocet s parametrom n =  0
Ukonci sa vykonavanie funkcie odpocet s parametrom n =  1
Ukonci sa vykonavanie funkcie odpocet s parametrom n =  2
Ukonci sa vykonavanie funkcie odpocet s parametrom n =  3
```

Rekurzia - úvod

Všimnite si, ako funguje rekurzívne volanie:

- volanie *odpocet(3)* počas behu volá *odpocet(2)*.
- v tomto momente **ostáva volanie** *odpocet(3)* **stáť a čaká**, kým sa skončí volanie *odpocet(2)* a až po skončení *odpocet(2)* bude volanie *odpocet(3)* **pokračovať!**
    - volanie *odpocet(2)* počas behu volá *odpocet(1)*.
    - v tomto momente **ostáva volanie** *odpocet(2)* **stáť a čaká**, kým sa skončí volanie *odpocet(1)* a až po skončení *odpocet(1)* bude volanie *odpocet(2)* **pokračovať!**
        - volanie *odpocet(1)* počas behu volá *odpocet(0)*
        - v tomto momente **ostáva volanie** *odpocet(1)* **stáť a čaká**, kým sa skončí volanie *odpocet(0)* a až po skončení *odpocet(0)* bude volanie *odpocet(1)* **pokračovať!**
            - volanie *odpocet(0)* **už nič ďalšie nevolá!!!**. Toto volanie je **špecifické v tom**, že sa pre parameter *n = 0* nevykoná rekurzívne volanie! Volanie *odpocet(0)* teda po výpise „Štart“ regulárne **skončí!**
        - po skončení *odpocet(0)* následne pokračuje *odpocet(1)* a skončí.
    - po skončení *odpocet(1)* následne pokračuje *odpocet(2)* a skončí
- po skončení *odpocet(2)* následne pokračuje *odpocet(3)* a skončí

Rekurzia

Všimnite si teda dôležitú vec, ako je naprogramovaná funkcia s rekurziou:

```
def odpocet(n):
    if n <= 0:
        print("Start!")
    else:
        print(n)
        odpocet(n-1)
```

Tzv. ukončujúca podmienka (base-case). Nastane v prípade, že n <= 0. Bez tejto ukončujúcej podmienky by sa rekurzia vykonávana teoreticky DONEKONEČNA!

Tzv. rekurzívna časť! Tu je rekurzívne volanie!

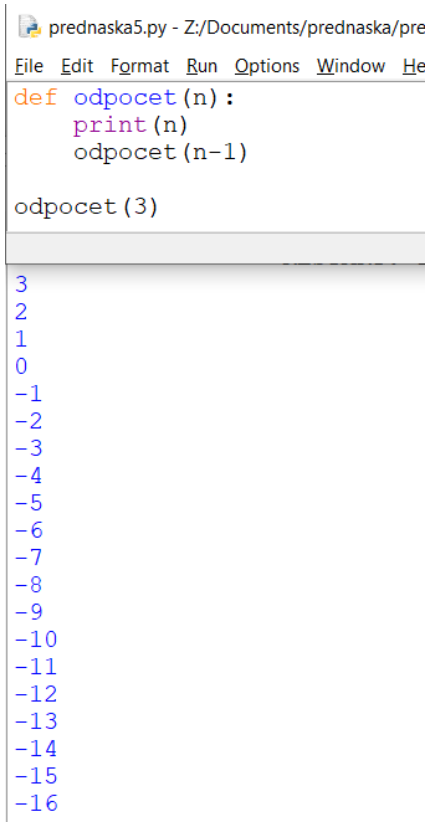
Každá funkcia, ktorá používa rekurziu, teda musí niekde obsahovať 2 časti:

a) rekurzívne volanie – funkcia volá samú seba, spravidla s **inou hodnotou, než má jej vstupný parameter!**

b) ukončujúca podmienka (angl. base-case) – časť kódu, ktorá **nepoužíva rekurzívne volanie**, vďaka ktorej sa rekurzia **zastaví!**

Rekurzia

Čo by sa stalo, ak by sme v našom kóde nemali ukončovaciu podmienku???



Program sa začne vnárať rekurzívne stále hlbšie a hlbšie…

Volanie *odpocet(3)* volá *odpocet(2*) a čaká kým skončí,
Volanie *odpocet(2)* volá *odpocet(1*) a čaká kým skončí,
Volanie *odpocet(1)* volá *odpocet(0*) a čaká kým skončí,
Volanie *odpocet(0)* volá *odpocet(-1*) a čaká kým skončí,
Volanie *odpocet(-1)* volá *odpocet(-2*) a čaká kým skončí,
Volanie *odpocet(-2)* volá *odpocet(-3*) a čaká kým skončí,
Volanie *odpocet(-3)* volá *odpocet(-4*) a čaká kým skončí,
Volanie *odpocet(-4)* volá *odpocet(-5*) a čaká kým skončí,
Volanie *odpocet(-5)* volá *odpocet(-6*) a čaká kým skončí,
Volanie *odpocet(-6)* volá *odpocet(-7*) a čaká kým skončí,
…..
až kým nám Python nepovie, že stačilo :)

Rekurzia

Python nám hovorí, že sme sa rekurzívne vnorili už príliš veľa krát:

```
-1015
-1016Traceback (most recent call last):
  File "Z:/Documents/prednaska/prednaska5.py", line 5, in <module>
    odpocet(3)
  File "Z:/Documents/prednaska/prednaska5.py", line 3, in odpocet
    odpocet(n-1)
  File "Z:/Documents/prednaska/prednaska5.py", line 3, in odpocet
    odpocet(n-1)
  File "Z:/Documents/prednaska/prednaska5.py", line 3, in odpocet
    odpocet(n-1)
  [Previous line repeated 1016 more times]
  File "Z:/Documents/prednaska/prednaska5.py", line 2, in odpocet
    print(n)
RecursionError: maximum recursion depth exceeded
```

Nekonečná rekurzia

Preto si teda dávajte **POZOR NA NEKONEČNÚ REKURZIU!!!**

Vždy je nutné dobre si zadefinovať **ukončovaciu podmienku!**

Rekurzia - využitie

- Prvý spôsob využitia rekurzie je pri implementácii matematických funkcií, ktoré sú definované rekurzívne.
- Typickým predstaviteľom matematickej funkcie, ktorá má rekurzívnu definíciu je **faktoriál.**
- Faktoriál čísla *n,* označený n!, je definovaný pre nezáporné čísla *n* nasledovným spôsobom:
  - ak $n = 0$, potom $0! = 1$
  - ak $n > 0$, potom $n! = n * (n-1)!$
- Všimnite si, že v tomto prípade už samotná definícia je vynikajúci recept na implementáciu funkcie, ktorá bude počítať faktoriál čísla *n*!

Rekurzia - využitie

- Faktoriál čísla *n,* označený n!, je definovaný nasledovným spôsobom:
  - ak *n* = *0*, potom 0! = 1
  - ak *n* > *0*, potom *n*! = *n* * (*n*-1)!

```
def faktorial(n):
    if n == 0:      #ukoncovacia podmienka
        return 1
    else:           #rekurzivna cast
        rekurzia = faktorial(n-1)
        vysledok = n * rekurzia
        return vysledok
```

Rekurzia - využitie

- V matematike platí:
  - 0! = 1
  - 1! = 1*0! = 1*1 = 1
  - 2! = 2*1! = 2*1*0! = 2*1*1 = 2
  - 3! = 3*2! = 3*2*1! = 3*2*1*0! = 3*2*1*1 = 6
  - 4! = 4*3! = 4*3*2! = 4*3*2*1! = 4*3*2*1*0! = 4*3*2*1*1 = 24
  - atď.

Rekurzia - využitie

- Ako sa bude správať náš program pre volanie napr. *faktorial(3)*
- volanie *faktorial(3)* zavolá funkciu *faktorial(2)*, ktorej návratovú hodnotu chce vložiť do svojej kópie premennej *rekurzia*.
  - volanie *faktorial(2)* zavolá funkciu *faktorial(1)*, ktorej návratovú hodnotu chce vložiť do svojej kópie premennej *rekurzia.*
    - volanie *faktorial(1)* zavolá funkciu *faktorial(0)*, ktorej návratovú hodnotu chce vložiť do svojej kópie premennej *rekurzia.*
      - volanie *faktorial(0)* vráti hodnotu 1 (pretože to má priamo definované vo svojej **ukončovacej podmienke**).
    - volanie *faktorial(1)* do *rekurzia* priradí *faktorial(0)* a teda vráti 1*1 = 1
  - volanie *faktorial(2)* do *rekurzia* priradí *faktorial(1)* a teda vráti 2*1 = 2
- volanie *faktorial(3)* do *rekurzia* priradí *faktorial(2)* a teda vráti 3*2 = 6

Rekurzia

- Ako je vidieť, správne naprogramovanie rekurzie predpokladá:
  - že rozumiete a viete korektne pracovať s návratovou hodnotou funkcie – všimnite si, že v prípade funkcie *faktorial(n)* vezmeme návratovú hodnotu volania *faktorial(n-1)* a ďalej s ňou pracujeme!
  - že rozumiete a viete korektne pracovať s parametrami a argumentami funkcie – všimnite si, že pri volaní funkcie *faktorial(n-1*) sa ako argument vloží hodnota výrazu *n-1*, kde *n* je hodnota *parametra* volania *faktorial(n)*

Rekurzia – globálne premenné

- **POZOR!** Často si študenti a študentky uľahčujú život a vytvoria rekurziu, ktorá síce naoko vypočíta správnu hodnotu, avšak používa pritom **globálne premenné.**
- Cieľom na PROG1 je pochopiť a naučiť sa, ako rekurzia funguje a ako navrhnúť funkciu, ktorá s ňou pracuje – čo sa Vám nepodarí, ak budete používať globálne premenné.
- Navyše, za rekurziu, ktorá používa globálne premenné, bude na skúške/teste **nula bodov!**

Rekurzia – na čo je to teda dobré???

- Rekurzia je silný programátorský nástroj na algoritmické riešenie inštancie nejakého problému, pričom táto inštancia sa dá vyriešiť tak, že:
  - 1) najprv vyriešim nejakú **menšiu inštanciu** toho istého problému
  - 2) následne vezmem toto riešenie menšej inštancie a z neho viem odvodiť riešenie pôvodného problému
  - 3) existuje nejaká malá inštancia daného problému, ktorej riešenie viem hneď určiť (sem patrí ukončovacia podmienka)

Rekurzia – na čo je to teda dobré???

- Vezmime si faktoriál
  - 1) najprv vyriešim nejakú **menšiu inštanciu** toho istého problému:

    Faktoriál čísla n vypočítam tak, že najprv vypočítam faktoriál n-1
  - 2) následne vezmem toto riešenie menšej inštancie a z neho viem odvodiť riešenie pôvodného problému

    Keď poznám faktoriál n-1, tak faktoriál n určím jednoducho ako súčin n-krát-faktoriál n-1
  - 3) existuje nejaká malá inštancia daného problému, ktorej riešenie viem hneď určiť.

    Faktoriál 0 je definovaný ako 1.

Rekurzia – na čo je to teda dobré???

- Keď mám teda programovať rekurziu, vždy si musím premyslieť:
  - 1) Aká je v danom prípade ukončovacia podmienka
  - 2) Ako vyriešiť daný problém tak, že **predpokladám,** že **mám riešenie menšieho problému** a **ako z neho budem vychádzať** pri riešení **pôvodného problému.**

Rekurzia – ako na rekurziu

- Niekedy je použitie rekurzie zrejmé, pretože problém je zadaný rekurzívne.
- Napríklad faktoriál je definovaný ako:
  - $n! = n*(n-1)!$ (rekurzívny krok)
  - $0! = 1$ (ukončovacia podmienka)
- Hľadanie NSD (angl. GCD) pomocou Euklidovho algoritmu definované ako:
  - GCD(a, b) = GCD(b, a%b) (rekurzívny krok)
  - GCD(a, 0) = a (ukončovacia podmienka)

Rekurzia – ako na rekurziu

- Inokedy to nie je priamo zadané, ale rekurziu pri riešení daného problému treba „objaviť“.
- Základné otázky sú:
  - „Viem riešiť daný problém tak, že budem vychádzať z riešenia **menšej inštancie** toho istého problému ? (rekurzívny krok)
  - Viem jednoducho určiť riešenie nejakej **základnej** / **najmenšej inštancie** toho istého problému? (ukončovacia podmienka) **(prípadne viacerých základných inštancií – ukončovacích podmienok môže byť aj viacero!)**

Rekurzia – ako na rekurziu

- Faktoriál:
  - „Viem riešiť daný problém tak, že budem vychádzať z riešenia **menšej inštancie** toho istého problému ? (rekurzívny krok)
    - Áno! Faktoriál *n*! viem ľahko vypočítať, ak poznám (*n*-1)! a ten len vynásobím číslom *n*, t.j. *n*! = *n**(*n*-1)!
  - Viem jednoducho určiť riešenie nejakej **základnej** / **najmenšej inštancie** daného problému? (ukončovacia podmienka)
    - Áno! Pre faktoriál existuje základná inštancia, faktoriál nuly, ktorá má hodnotu 1, 0! = 1.

Rekurzia – príklad

Definujte funkciu *sucet(n)*, ktorá má vstupný parameter kladné celé číslo *n*. Funkcia načíta *n* čísiel z klávesnice a vráti ich súčet. Zamyslite sa, ako riešiť takúto úlohu **rekurzívne!**

Najprv si ujasnime, čo má funkcia robiť, najmä, čo má vracať:

1) Ak funkciu zavolám s argumentom *n*, znamená to, že počítač načíta *n* čísiel z klávesnice a volanie vráti súčet načítaných čísiel.

2) Teda o **každom volaní** *sucet(n)* viem povedať, že:
- načíta *n* čísiel z klávesnice
- vráti ich súčet

Rekurzia – príklad

Položme si prvú otázku, ktorá vedie na rekurzívne riešenie daného problému:

1) „Viem riešiť daný problém tak, že budem vychádzať z riešenia **menšej inštancie** toho istého problému ?

Čo je náš problém? ---> Nájsť súčet n čísiel zadaných z klávesnice.
*Čo by bola **menšia inštancia problému?** ---> Napríklad súčet (n-1) čísiel načítaných z klávesnice.*

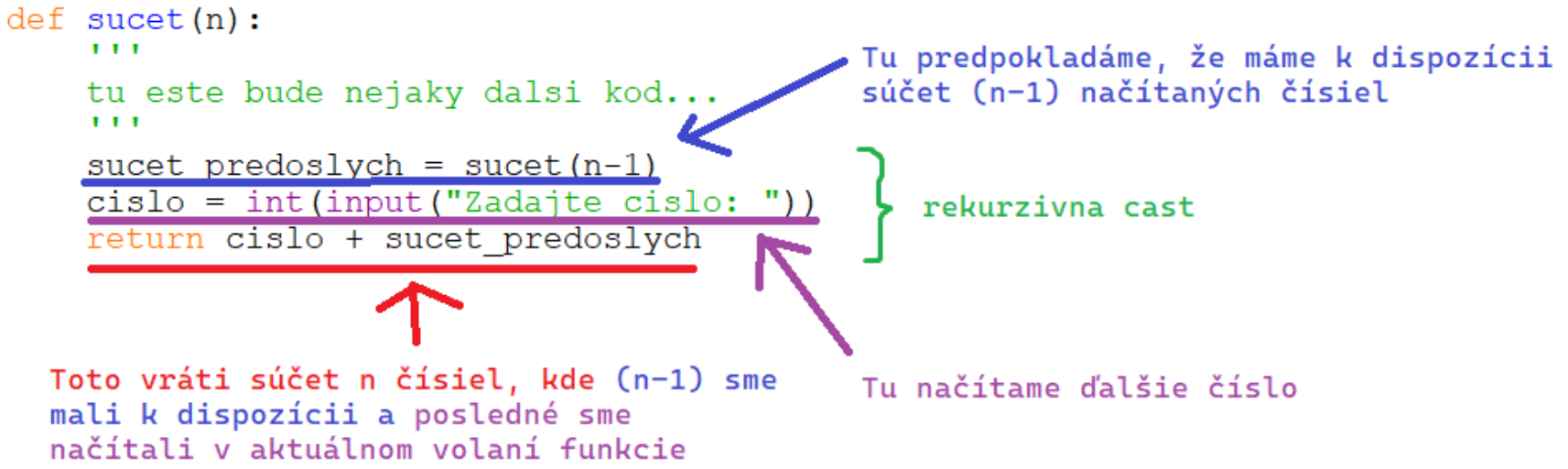
*Viem riešiť problém súčtu n čísiel zadaných z klávesnice, ak **predpokladám**, že už mám k dispozícii súčet (n-1) čísiel, ktoré boli zadané z klávesnice???*

Ako získam súčet (n-1) čísiel načítaných z klávesnice???
*---> **No predsa volaním sucet(n-1)***

Rekurzia – príklad

*Viem riešiť problém súčtu n čísiel zadaných z klávesnice, ak **predpokladám**, že už mám k dispozícii súčet (n-1) čísiel, ktoré boli zadané z klávesnice???*

*ÁNO!!! Stačí, ak vezmem súčet (n-1) čísiel, ktorý už mám k dispozícii a len k nemu pripočítam načítané číslo z klávesnice! Tým **musím dostať** súčet n čísiel zadaných z klávesnice.*



Rekurzia – príklad

Položme si druhú otázku, ktorá vedie na určenie ukončovacej podmienky:
2) Viem jednoducho určiť riešenie nejakej **základnej** / **najmenšej inštancie** daného problému?

Čo je náš problém? ---> Nájsť súčet n čísiel zadaných z klávesnice.

*Čo by bola **základná / najmenšia inštancia problému?** ---> Súčet 1 načítaného čísla, pretože to znamená, že výsledný súčet je **samotné načítané číslo!***

Teda v prípade, že sa funkcia sucet(n) volá so vstupným argumentom 1, čiže parameter n == 1, tak funkcia vráti priamo načítané číslo!.

*Tým sme určili **ukončovaciu podmienku** rekurzie.*

Rekurzia – príklad

```
def sucet(n):
    if n == 1:
        cislo = int(input("Zadajte cislo: "))
        return cislo
    else:
        sucet_predoslych = sucet(n-1)
        cislo = int(input("Zadajte cislo: "))
        return cislo + sucet_predoslych
```

Ukončovacia podmienka (base-case)

Rekurzívna časť