# Základy programování v jazyce Python

## Lekce 6: abstrakce - funkce

# Opakování - Slovníky, seznamy a řetězce

Zadání všech cvičení najdete ve studijních materiálech

# Motivace

- Při psaní delších programů se menší bloky budou opakovat.
- Počítači je jedno, jestli se kód opakuje, lidem ne - čitelnost kódu je důležitá.
- Funkce představují abstrakci nad kódem:

```
cisla = [12, 231, 51, 624, 12, 76]
```

Kód:

```
suma = 0
for i in cisla:
    suma = suma + i
```

Abstrakce:

```
suma = scitej(cisla)
```

# Příklad ze života

Chcete zkombinovat prvky dvou seznamů tak, abyste měli seznam párů:

```
['a', 'b', 'c']
['A', 'B', 'C']
```

```
[ ('a', 'A'), ('b', 'B'), ('c', 'C') ]
```

Proč byste to chtěli?

Například Vám to umožní vytvořit slovník jednoduše typovým převodem:

```
slovnik = dict(seznam_paru)
```

# Řešení

```
male = ['a', 'b', 'c']
velke = ['A', 'B', 'C']

male_velke = []
for i in range(len(male)):
    par = (male[i], velke[i]) # n-tice (tuple)
    male_velke.append(par)
```

Kód je krátký a přehledný.

**Ale co když chcete udělat stejnou věc pro čísla a písmena?**

# Řešení

```
cisla = [1, 2, 3]
male = ['a', 'b', 'c']

cisla_male = []
for i in range(len(cisla)):
    par = (cisla[i], male[i]) # n-tice (tuple)
    cisla_male.append(par)
```

Napsali jsme v podstatě stejný kód a jenom vyměnili názvy proměnných.

Ještě horší by bylo, kdybyste chtěli páry párů:
`[ (1, ('a', 'A')), ...]`

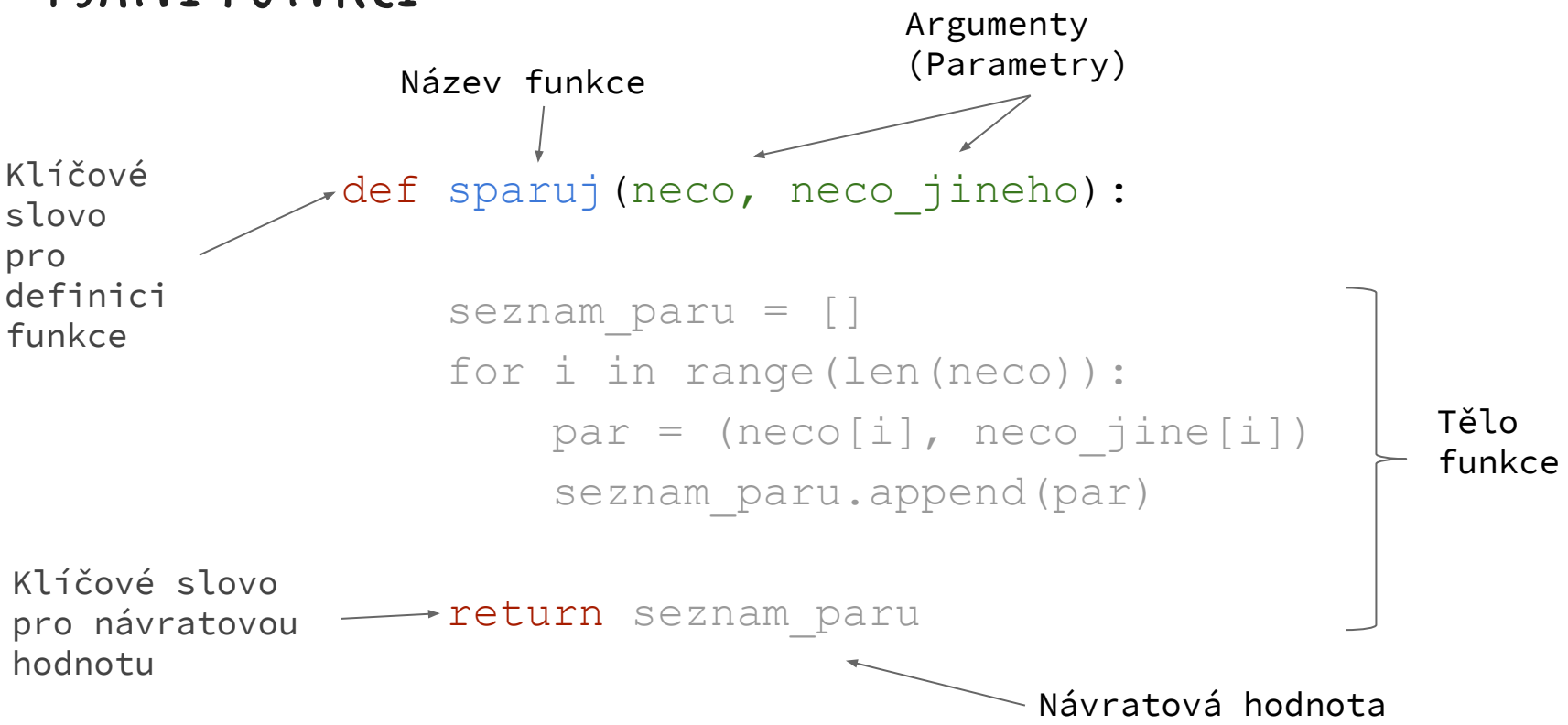
# Abstrakce

- Snažíte se spárovat dva seznamy.
- Nezajímá Vás jak.

```
seznam_párů = sparuj(neco, neco_jineho)
```

Už tu abstrakci musí jenom někdo napsat...

# Psaní funkcí:




Mimochodem: funkce sparuj je natolik užitečná, že v Pythonu již existuje pod názvem **zip**

# Rámec proměnné

Funkce mají svoje lokální proměnné:

```
def obal_slovem(neco, slovo):
    g = str(neco)
    return slovo + g + slovo

g = 2
print(obal_slovem(g, 'dust'))
print(type(g)) # neni str
```

Po **return** to **g** nepřežije.

není stejná proměnná!

Globální proměnná **g** se nezměnila použitím **g** ve funkci.

# DRY vs WET

Don't Repeat Yourself

VS

Write Everything Twice
We Enjoy Typing
Waste Everyone's Time

# Cvičení

Napište funkci, která vrátí řetězec:

"Byl jsem tady!"

Tuhle důležitou funkci vhodně pojmenujte.

```
def nazev():
    return navratova_hodnota

hodnota = nazev()
```

# Cvičení 1 - rámeček

Přepište rámovací cvičení ze druhé hodiny na funkci.

Úkolem bylo obalit řetězec libovolné délky vhodně dlouhým rámečkem:

# Cvičení 2 - změna seznamu

Napište funkci, která přidá prvek do již existujícího seznamu. Pokud se prvek v seznamu nachází, funkce vypíše, že již je prvek obsažený a prvek nepřidá.

```
seznam = [] # Prázdný seznam
pridej(seznam, 1) # OK
pridej(seznam, 2) # OK
pridej(seznam, 1) # Vypíše "prvek 1 už v seznamu je!"
```

# Slovo return

Pokud jste napsali funkci **pridej** s klíčovým slovem **return**, zkuste následující kód:

```
s = []
s2 = pridej(s, 1)
pridej(s2, 2)

print(s2)
print(s)
```

Seznamy můžete měnit -> mutable objekty změněné ve funkci se změní i mimo ni -> return je zbytečné.

Poznámka: Pokud nenapíšete return, funkce vrátí None.
Poznámka 2: Všechno, co se nachází ve funkci za return, se nevykoná.

# Otázka

Co udělá následující program:

```
def pravdepodobne(veta):
    veta = veta + ', pravděpodobně.'

a = 'Programováním v Pythonu neohrožujete svoje zdraví'
pravdepodobne(a)
```

# Cvičení 3 - unikátnost

Napište funkci, která zkontroluje, jestli se hodnoty ve slovníku vyskytují jenom jednou.

Funkce vrátí **True** nebo **False**.

# Cvičení 4 - inverze

Napište funkci, která invertuje slovník. To znamená, že z hesel budou hodnoty a z hodnot budou hesla.

Pak přidejte podmínku, která zavolá funkci z předcházejícího cvičení. Pokud vrátí **False**, tak inverze neproběhne a vypíše se varování.

# Cvičení 5 - kámen, nůžky, papír

*Pozn.: Funkce si průběžně testujte!*

Napište funkci, která náhodně vrátí 'kámen','nůžky' nebo 'papír'.
Můžete využít funkci **choice** z balíčku **random**. Podívejte se online např. sem

Pak vytvořte funkci, která vezme dvě slova a vrátí -1, 0, nebo 1 (prohra, remíza, výhra).

Napište funkci, která s Vámi bude hrát kámen, nůžky, papír.  To znamená:

```
kamen_nuzky_papir('papir') # vyhra / prohra
kamen_nuzky_papir('kulomet') # chybné gesto rukou
```

Funkci napište tak, aby volala dvě předošlé funkce.

Dále vylepšete hru použitím funkce input.

Na závěr napište funkci, která bude počítat skóre a zastaví se, až Vy nebo počítač dosáhnete 3 bodů.

# Cvičení 5 - kámen, nůžky, papír - ještěr, Spock

Nyní budeme modifikovat hru kámen, nůžky, papír o volby ještěr, Spock (obr).

*As Sheldon explains, "Scissors cuts paper, paper covers rock, rock crushes lizard, lizard poisons Spock, Spock smashes scissors, scissors decapitates lizard, lizard eats paper, paper disproves Spock, Spock vaporizes rock, and as it always has, rock crushes scissors."*

Zde se ukáže jak moc jste si hezky napsali předchozí hru kámen nůžky papír. Pokud jste si poctivě vypsali všechny možnosti ručně (celkem 9 - pro kámen, nůžky, papír) tak teď byste museli mít podmínek 25. Zkuste se zamyslet jak tento problém obejít.

PS.: Třeba pomocí předdefinovaného slovníku, ale možnostem se meze nekladou.

A když už v tom budete tak zároveň zajistěte, aby se vypsala vždy správná vazba a jestli jste vyhráli, např.: knpjs('rock','spock') -> *Spock vaporizes rock - you have lost!*

# Dokumentování funkcí

*“Indeed, the ratio of time spent reading versus writing is well over 10 to 1. We are constantly reading old code as part of the effort to write new code. ...[Therefore,] making it easy to read makes it easier to write.”*

- Robert C. Martin

# Dokumentování funkcí

Funkce můžete dokumentovat řetězcem po prvním řádku. Pro dlouhý text použijte "troje dvojité" uvozovky.

```
def annoy():
    """Using this function will drive you insane.
    Srsly!
    Don't use this.
    """
    var = choice([i for i in globals() if i[0] != '_'])
    print('Ooops, {} is gone!'.format(var))
    globals()[var] = "You didn't really need this, did you?"
```

# Dokumentování funkcí

Tento řetězec se zobrazí při zavolání funkce **help** nebo použití **?** v jupyteru.

```
help(annoy)
?annoy
annoy?
```

# Jak psát nečitelně?

nicneříkající název funkce a parametrů
programátor si asi chtěl ušetřit psaní

```
def f1(p1, p2, p3=2):
    """Fakt se to zobrazi, kdyz dam v Jupyteru otaznik???
    A lame to radky??
    Paraaaaada!
    """
    p1 = p1.strip()
    p1 = p2+p1+p2
    try:
        p1 = float(p1)
        return p1*p3
    except ValueError:
        return p1
```

nesmyslný a tím pádem zbytečný docstring

obtížně pochopitelný kód - tomu se
ale někdy nedá pomoct

# Jak psát čitelně?

popisný název funkce a parametrů. Volíme angličtinu, protože nevíme, kdo všechno to po nás bude číst.

```
def wrap_and_multiply(main_string, wrapper_string, factor=2):
    """
    """
    main_string = main_string.strip()
    wrapped_string = wrapper_string+main_string+wrapper_string
    try:
        wrapped_num = float(wrapped_string)
        return wrapped_num*factor
    except ValueError:
        return wrapped_string
```

Podrobný a strukturovaný doctring, zoom na další straně.

pochopitelné a popisné názvy proměnných, i za cenu toho, že jsme jich definovali víc.

# Jak psát čitelně? Docstrings

začíná a končí "trojími dvojitými" uvozovkami.

```
"""
Strip the main_string from whitespaces, wrap it in
wrapper_string and try to multiply by factor. If the
wrapped string cannot be converted to a number,
return the wrapped string without multiplying.

Args:
    main_string: string with a numeral, may include
                 leading and trailing whitespaces
    wrapper_string: string for wrapping, possibly
                    containing a numeral. No processing
                    of this string is performed in the function!
    factor: multiplication factor - should be a number

Returns:
    if wrapped string can be converted to a number:
        wrapped string converted to number
        multiplied by the factor
    else:
        wrapped string
"""
```

Stručné vysvětlení toho, co funkce dělá, slovesa jsou ve formě příkazu (return), ne popisu (returns).

seznam argumentů s vysvětlením, co funkce očekává

co funkce vrací

# Cvičení

Zdokumentujte všechny funkce, které jste dnes napsali.

# K samostudiu (nebude na zkoušce)

# Cvičení 6 - nejdelší řetězec

Napište funkci, která vrátí delší ze dvou řetězců:

```
delsi('nejnezapamatovávatelnější',
      'trichlor(bischlorfenyl)ethan')
# 'trichlor(bischlorfenyl)ethan'
```

# Víc o funkcích -- argumenty

V předcházející funkci by se hodilo mít libovolný počet argumentů.

Vyzkoušejte následující funkce, rozmyslete si co dělají:

```
def vypis_arg(*arg):
    print(arg)
    print(type(arg))


def vypis_arg2(prvy, *arg):
    print(prvy)
    print(arg)
```

# Výchozí parametry

Funkce mohou být "přednastaveny":

```
def moje_kočky(kolik=1, barva='černé'):
    kočky = 'kočku'
    if kolik > 5 or kolik == 0:
        kočky = 'koček'
    elif kolik > 1:
        kočky = 'kočky'
    return ' '.join(['Mám', str(kolik), kočky, barva,
'barvy'])

moje_kočky() # 'Mám 1 kočku černé barvy'
moje_kočky(4) # 'Mám 4 kočky černé barvy'
moje_kočky(12, 'zelené') # 'Mám 12 koček zelené barvy'
```

# Pořadí argumentů nemusí nic znamenat…

Zatím jsme využívali tzv. poziční parametry v našich funkcích.

Python taky dovoluje použít názvy parametrů namísto (nebo v kombinaci s) pořadí:

```
moje_kočky(barva='zelené', kolik=12)
# 'Mám 12 koček zelené barvy'
```

# Libovolný počet koček

Funkci můžeme předělat ať vezme víc typů koček.

```
def moje_kočky(*arg):
    veta = ['Mám']
    for i,j in arg:
        kočky = 'kočku'
        if i > 5 or i == 0:
            kočky = 'koček'
        elif i > 1:
            kočky = 'kočky'
        veta.extend([str(i), kočky, j, 'barvy,'])

    return ' '.join(veta)

moje_kočky((1,'zelené'), (12, 'černé'))
```

# Kombinování pozičních parametrů a klíčových slov

```
def moje_kočky(*arg, sloveso='Mám'):
    veta = [sloveso]
    for i,j in arg:
        kočky = 'kočku'
        if i > 5 or i == 0:
            kočky = 'koček'
        elif i > 1:
            kočky = 'kočky'
        veta.extend([str(i), kočky, j, 'barvy,'])

    return ' '.join(veta)
```

# Kombinování pozičních parametrů a klíčových slov

Při volání funkce musí být poziční argumenty před klíčovými slovy.

```
moje_kočky((1,'zelené'), sloveso='Večeřím') # OK
moje_kočky(sloveso='Večeřím', (1,'zelené')) # SyntaxError
```