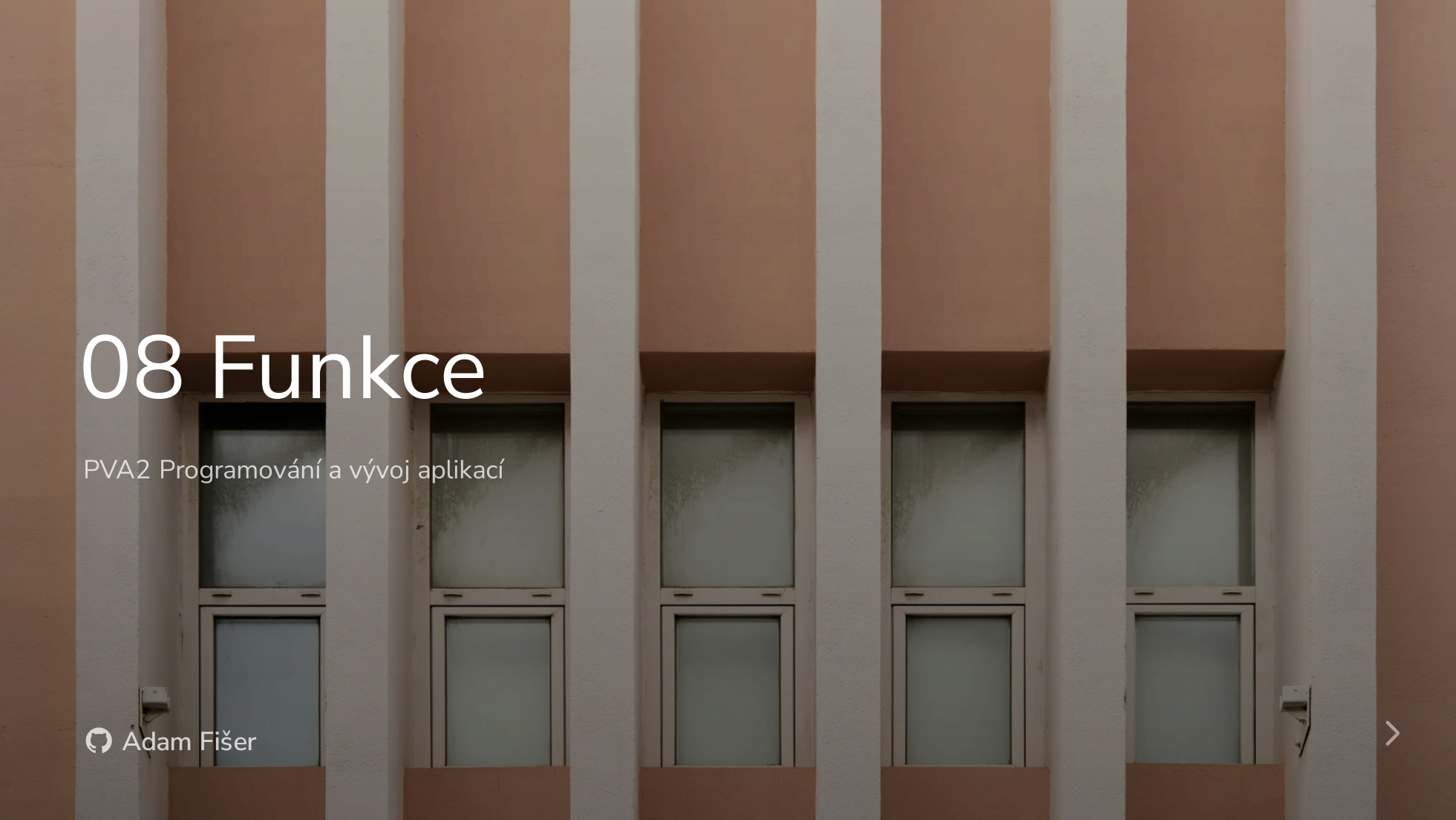

08 Funkce

PVA2 Programování a vývoj aplikací

Adam Fišer

Obsah

Využití funkce

- Funkce jsou vhodný způsob jak rozdělit kód do užitečných bloků.
- Zlepšuje přehlednost a čitelnost zdrojového kódu.
- Umožňuje opakovaně využít část kódu.
- Umožňuje předávat parametry a získávat návratové hodnoty.

Deklarace funkce

- Pro definici funkce se využitá klíčové slovo `def`,
- následuje název funkce a v závorce seznam formálních parametrů. Parametry mohou být i prázdné.
- Deklarace je ukončena dvojtečkou.
- Příkazy tvoří tělo funkce, začínají na dalším řádku a musí být odsazeny.

```
1   # funkce s názvem mojeFunkce
2   def mojeFunkce():
3       # Kód musí být odsazen
4       return "Ahoj"
```

Návratová hodnota

- Funkce může vracet hodnotu pomocí klíčového slova `return`.
- Pokud funkce nemá `return`, vrací `None`.
- Po `return` se může vyskytovat pouze jedna hodnota.
- Po `return` se funkce ukončí a další příkazy se nevykonají.
- Vrácenou hodnotu můžete použít k přiřazení do proměnné nebo ji jen vypsat.

```
# funkce s názvem mojeFunkce
def mojeFunkce():
    # Kód musí být odsazen
    return "Ahoj"
    print("Tento řádek se nevykoná")
```

```
def sumTwoValues(a, b):
  return a + b

# Do proměnné c přiřazen výsledek funkce sumTwoValues
c = sumTwoValues(3, 12)
```

Volání funkce

- Funkce se volá pomocí jejího názvu a závorek.
- Pokud funkce přijímá parametry, je nutné je předat v závorkách.
- Pokud funkce vrací hodnotu, je možné ji uložit do proměnné.

```
# funkce s názvem mojeFunkce
def mojeFunkce():
    # Kód musí být odsazen
    return "Ahoj"
```

```
# Volání funkce přes název
mojeFunkce() # Volání funkce

var = mojeFunkce() # Volání funkce, výsledek uložen do proměnné var

print( mojeFunkce() ) # Vypíše Ahoj
```

Volání funkce

```
# funkce s názvem mojeFunkce
def mojeFunkce():
    # Kód musí být odsazen
    return "Ahoj"
```

```
# zavoláme funkci pětkrát
for i in range(5):
    mojeFunkce()
```

Parametry funkce

- Parametry jsou hodnoty, které funkce přijímá a jsou nutné pro vykonání funkce.
- Parametry jsou uvedeny v závorce a odděleny čárkou.
- Parametry mají pouze lokální platnost.
- Můžeme nastavit výchozí hodnotu parametru přiřazení hodnoty za `=`

```
1  def pozdrav(jmeno):
2      return "Ahoj " + jmeno
3
4  print ( pozdrav("Adam") ) # Ahoj Adam
```

Parametry funkce

```
1  def mojeFunkce(parameter, secondParam):
2    return parameter + " " + secondParam
3
4  mojeFunkce("argument") # vrátí chybu TypeError
```

- Ve výchozím nastavení musí být funkce volána se správným počtem parametrů. Pokud funkce očekává 2 parametry, musíte ji zavolat se 2 argumenty.
- nebo můžeme využít výchozí hodnotu parametru

Výchozí hodnota parametru

- Můžeme nastavit výchozí hodnotu parametru přiřazení hodnoty za `=`
- Pokud není parametr předán, použije se výchozí hodnota.
- Parametry se přebírají v pořadí, není-li určeno jinak.

```
# Výchozí hodnota parametru
def pozdrav(jmeno = "UN"):
    return "Ahoj " + jmeno

print ( pozdrav() ) # Ahoj UN
```

```
def multiply(a, b=2, c=1):
    return a * b + c

multiply(3, 14, 10) # Všechny argumenty

multiply(3) # Jeden povinný argument a

multiply(3, 14) # Předány dva argumenty a, b
multiply(3, c=10) # Předány dva argumenty a, c
```

Klíčová slova jako argumenty

- Pokud chcete předat argumenty do funkce, ale nechcete si pamatovat pořadí, můžete použít klíčová slova `kwArg = hodnota`
- Klíčová slova musí být uvedena po pozicích argumentů.
- Klíčová slova mohou být v libovolném pořadí.

```
1   def cat(food, state='stále hladová', action='mňau'):
2       print("-- Tahle kočka by nechtěla", action, end=' ')
3       print("kdybyste ji dali", food)
4       print("-- krásná srst", breed)
5       print("-- To je", state, "!")
6
7
8   cat('kuře')                       # 1 argument dle pozice
9   cat(food='kuře')                  # 1 argument klíčové slovo
10  cat(food='rybu', action='kousat')  # 2 kwarg
11  cat(action='kousat', food='rybu')  # 2 kwarg
12  cat('beef', 'šťastná', 'syčet')    # 3 poziční argument
13  cat('objetí', state='vrní')        # 1 poziční, 1 klíčové slovo
```

Side effect

- Funkce je bez tzv. **vedlejších efektů** (side effect), tj. používá pouze své parametry a nepoužívá žádné proměnné definované mimo ni (např. vstup od uživatele). Stejně tak mimo návratové hodnoty nijak neovlivňuje běh programu.
- Funkci bez vedlejších efektů se říká čistá funkce (pure function). Její výhodou je, že pro stejný vstup vždy vrací stejný výstup, což například usnadňuje testování nebo hledání chyby.

```
1  # Špinavá funkce
2  # čte proměnnou "zvenku".
3  # Může tedy v různých situacích vracet různé výsledky.
4
5  exchange_rate = 26
6  def convert_to_euro(crown):
7      return crown * exchange_rate
```

```
1  # Takto uvedená funkce je již čistou funkcí.
2  def convert_to_euro(crown, exchange_rate):
3      return crown * exchange_rate
```

Rekurze

- Rekurze je technika, kdy funkce volá sama sebe.
- Rekurze je užitečná pro řešení problémů, které lze rozdělit na menší části.
- Rekurze může být náročná na paměť, pokud je hloubka rekurze příliš velká.
- Rekurze musí mít podmínku ukončení, aby se zabránilo nekonečnému volání.

```
1  def factorial(n):
2      if n == 1:
3          return 1
4      else:
5          return n * factorial(n-1)
```

Děkuji za pozornost

Otázky?

Repository / Prezentace