
KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI
www.inf.upol.cz

Jan Laštovička
https://www.inf.upol.cz/lide/jan-lastovicka
jan.lastovicka@upol.cz
17. listopadu 12, 771 46 Olomouc

Úvod do programovacích stylů ◇ poznámky k přednášce

1. Procedurální programování

verze z 26. září 2024

1 Úvod

Cílem kurzu je seznámit se s různými programovacími styly. Díky tomu dokážeme zvolit vhodný styl pro řešení problému. Volbou vhodného stylu se výrazně zlepší čitelnost kódu.

V kurzu budeme používat programovací jazyk Python (`https://www.python.org`) a vývojové prostředí IDLE, které je součástí standardní instalace Pythonu.

2 Procedury

Začneme opakováním procedurálního stylu programování. Funkcím v Pythonu budeme v této přednášce říkat **procedury**.

Například uvažujme proceduru `factorial`:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

Procedura je dána svými **parametry** a **tělem**. Například procedura `factorial` má jediný parametr `n` a tělo tvoří jediný příkaz podmíněného vykonávání `if`. Proceduru můžeme **zavolat**:

```
>>> factorial(5)
120
```

Proceduru voláme s hodnotami nazývanými **argumenty** a volání vrací **návratovou hodnotu**. Tedy například zavolání procedury `factorial` s argumentem `5` vrací číslo `120`.

Jméno není součástí procedury. Například táž procedura může mít různá jména:

```
>>> fact = factorial
>>> fact(5)
120
```

Jménem procedury myslíme název proměnné, která proceduru obsahuje. Například právě uvedená procedura má dvě jména: `factorial` a `fact`.

Procedura je podobně jako například číslo hodnota. Procedura tedy může být hodnotou proměnné, může být argumentem volání procedury a dokonce i návratovou hodnotou. Například procedura:

```
def apply_twice(procedure, n):
    return procedure(procedure(n))

>>> apply_twice(factorial, 3)
720
```

Zavolá zadanou proceduru `procedure` na návratovou hodnotu volání procedury `procedure` na `n`. V předchozím příkladě se spočítá faktoriál z faktoriálu tří.

3 Datové struktury

Hodnoty v Pythonu typu `list` nazýváme **pole**. Například `[1, 2, 3]` je pole obsahující tří prvky. **Datovou strukturou** můžeme reprezentovat polem. **Položky** datové struktury ztotožníme s prvky pole.

Například bod v rovině můžeme reprezentovat datovou strukturou `[`x`, `y`]`, kde položky určují $x$ovou a $y$ovou souřadnici bodu. **Typ datové struktury** pojmenovává struktury se shodnými typy položek. Například můžeme zavést typ struktur `point` pro struktury `[`x`, `y`]` s číselnými položkami.

Příklad práce s bodem:

```
>>> p1 = [3, 4]
>>> p2 = [p1[0] + 1, p1[1] + 2]
>>> p2
[4, 6]
```

Můžeme napsat proceduru na posun bodu:

```
def move_point(point, dx, dy):
    return [point[0] + dx, point[1] + dy]
```

Otestujeme:

```
>>> move_point(p1, 1, 2)
[4, 6]
```

Procedura, která vytváří datovou strukturu, se nazývá **konstruktor**. Konstruktor struktur typu *`type`* má vždy jméno `make_`*`type`*. Napíšeme si konstruktor bodu:

```
def make_point(x, y):
    return [x, y]
```

Procedury, které vrací položky datové struktury se nazývají **selektory**. Selektor položky *`propery`* má jméno `get_`*`type`*`_`*`property`*. Selektory souřadnic bodu:

```
def get_point_x(point):
    return point[0]

def get_point_y(point):
    return point[1]
```

Ukázka:

```
>>> p1 = make_point(3, 4)
>>> get_point_x(p1)
3
>>> get_point_y(p1)
4
```

Používání kontruktoru a selektorů zvyšuje čitelnost kódu. Starou verzi posunu bodu:

```
def move_point(point, dx, dy):
    return [point[0] + dx, point[1] + dy]
```

upravíme na:

```
def move_point(point, dx, dy):
    return make_point(get_point_x(point) + dx,
                      get_point_y(point) + dy)
```

4 Abstraktní datové struktury

Abstraktní datová struktura je dána pouze konstruktorem a selektory. Její uživatel neví, jakým způsobem je datová struktura reprezentovaná.

Bod můžeme definovat trojicí procedur:

- `make_point(`*x*`, `*y*`) => `*point*
- `get_point_x(`*point*`) => `*x*
- `get_point_y(`*point*`) => `*y*

Výhodou používání abstraktních datových struktur je snadná změna reprezentace datové struktury.

Představme si, že chceme změnit reprezentaci bodu z dvouprvkového pole na tříprvkové pole, kde první prvek pole bude řetězec `"point"` indikující, že se jedná o bod. Tedy:

`[`*x*`, `*y*`]` → `["point", `*x*`, `*y*`]`

Jediné, co je potřeba udělat, je změnit konstruktor a selektory:

```
def make_point(x, y):
    return ["point", x, y]

def get_point_x(point):
    return point[1]

def get_point_y(point):
    return point[2]
```

Ostatní kód není potřeba měnit.

Například:

```
def move_point(point, dx, dy):
    return make_point(get_point_x(point) + dx,
                      get_point_y(point) + dy)
```

Funguje beze změny:

```
>>> p1 = make_point(3, 4)
>>> p1
['point', 3, 4]
>>> p2 = move_point(p1, 1, 2)
>>> p2
['point', 4, 6]
```

Typový predikát je predikát, který rozhoduje, zda je hodnota datová struktura určitého typu. Typový predikát pro struktury typu *type* má jméno `is_`*type*.

Typový predikát bodu:

```
def is_point(value):
    return (type(value) == list
            and len(value) == 3
            and value[0] == "point")
            and type(value[1]) == int
            and type(value[2]) == int)
```

Test:

```
>>> is_point(p1)
True
>>> is_point([3, 4])
False
>>> is_point(3)
False
>>> is_point(["point", 3, "4"])
False
```

Říkáme, že procedura **nemá vedlejší efekt**, pokud jediný efekt jejího zavolání je vrácení hodnoty. Procedura bez vedlejšího efektu tedy nemůže měnit své argumenty.

Selektory přirozeně nemají vedlejší efekt. Konstruktor `make_point` sice nemá vedlejší efekt, ale jak uvidíme dále, obecně konstruktory vedlejší efekt mohou mít. Procedura `move_point` také nemá vedlejší efekt. Po zavolání vždy vrací nový bod:

```
>>> p1 = make_point(3, 4)
>>> p1
['point', 3, 4]
>>> p2 = move_point(p1, 1, 2)
```

```
>>> p2
['point', 4, 6]
>>> p1
['point', 3, 4]
```

5 Mutátory

Procedury, které mění položky datové struktury, nazýváme **mutátory**. Mutátor, který očekává datovou strukturu typu *type* a hodnotu položky *property*, má název `set_`*type*`_`*property*. Mutátory patří k definici abstraktní datové struktury a mají vedlejší efekt.

Mutátory bodu:

```
def set_point_x(point, x):
    point[1] = x

def set_point_y(point, y):
    point[2] = y
```

Příklad použití:

```
>>> p1 = make_point(3, 4)
>>> p1
['point', 3, 4]
>>> set_point_x(p1, 5)
>>> p1
['point', 5, 4]
>>> p2 = p1
>>> set_point_y(p1, 3)
>>> p1
['point', 5, 3]
>>> p2
['point', 5, 3]
```

Pomocí mutátorů můžeme posunout bod tak, že dojde k jeho změně:

```
def move_point(point, dx, dy):
    set_point_x(point, get_point_x(point) + dx)
    set_point_y(point, get_point_y(point) + dy)
```

Příklad:

```
>>> p1 = make_point(3, 4)
>>> move_point(p1, 1, 2)
>>> p1
['point', 4, 6]
```

V případě, že mají položky datové struktury výchozí hodnoty, můžeme upravit její konstruktor tak, že je uživatel zadávat nemusí.

Bod bude mít po vytvoření položky `x` a `y` rovny nule:

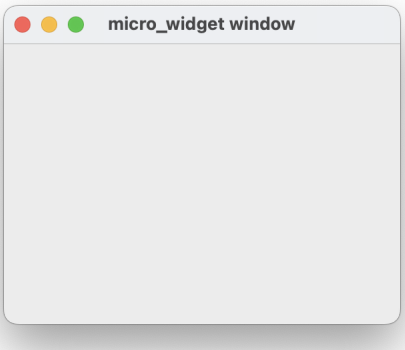
```
def make_point():
    return ["point", 0, 0]
```

Po vytvoření si bod můžeme posunout, kam potřebujeme:

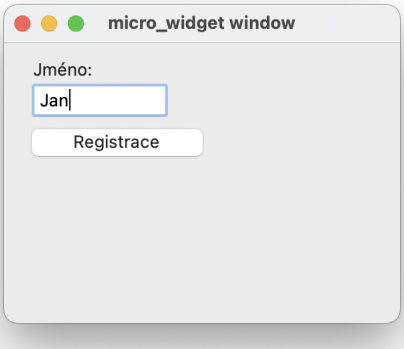
```
>>> p1 = make_point()
>>> p1
['point', 0, 0]
>>> move_point(p1, 3, 4)
>>> p1
['point', 3, 4]
```

6 Procedurální uživatelské rozhraní

Procedurální styl programování si vyzkoušíme na malé knihovně vytvářející grafické uživatelské rozhraní. Knihovna se jmenuje Micro Widget (`micro_widget`). Knihovnu můžeme použít k vytvoření grafického okna (`window`):



a vložení ovládacích prvků (`widgets`):



Knihovna nabízí tři typy ovládacích prvků. Můžeme vytvořit popisek (`label`), tlačítko (`button`) a textové pole (`entry`). Okna a ovládací prvky máme k dispozici ve formě abstraktních datových struktur, jejichž konstruktory mají vedlejší efekt v podobě otevření okna nebo vložení prvku do okna. Kromě základních procedur mají ovládací prvky i **destruktory**, které je odebírají z okna. Úplný popis knihovny naleznete v její referenční příručce nalézající se v souboru `micro_widget.pdf`.

Základní použití knihovny si ukážeme na vytvoření výše zobrazeného okna. Knihovna se nalézá v souboru `micro_widget.py`, který musí být ve složce s programem. Nejprve je nutné knihovnu importovat:

```
import micro_widget as mw
```

Otevřeme nové okno:

```
window = mw.display_window()
```

Do okna přidáme popisek, kterému nastavíme text a pozici:

```
label = mw.make_label(window)
mw.set_label_text(label, "Jméno:")
mw.set_label_x(label, 20)
mw.set_label_y(label, 10)
```

Přidáme textové pole:

```
entry = mw.make_entry(window)
mw.set_entry_x(entry, 20)
mw.set_entry_y(entry, 30)
```

a tlačítko:

```
button = mw.make_button(window)
mw.set_button_text(button, "Registrace")
mw.set_button_x(button, 20)
mw.set_button_y(button, 60)
```

Textové pole dáme do dat tlačítka:

```
 mw.set_button_data(button, entry)
```

Data budou k dispozici v proceduře volané při stisku tlačítka:

```
def button_click_handler(button):
    entry = mw.get_button_data(button)
    text = mw.get_entry_text(entry)
    print("Registrace uživatele:", text)
```

Na závěr nastavíme příkaz tlačítka:

```
mw.set_button_click_handler(button, button_click_handler)
```

Nyní zadáme `Jan` do textového pole a po kliku na tlačítko aplikace vytiskne:

```
Registrace uživatele: Jan
```

Výše uvedený příklad pouze demonstruje použití knihovny. Pro lepší čitelnost kódu si zavedeme datovou strukturu pro náš jednoduchý formulář. Začneme vytvořením pomocných procedur na vytvoření popisku:

```
def make_form_label(window):
    label = mw.make_label(window)
    mw.set_label_text(label, "Jméno:")
    mw.set_label_x(label, 20)
    mw.set_label_y(label, 10)
    return label
```

textového pole:

```
def make_form_entry(window):
    entry = mw.make_entry(window)
    mw.set_entry_x(entry, 20)
    mw.set_entry_y(entry, 30)
    return entry
```

a tlačítka:

```
def make_form_button(window):
    button = mw.make_button(window)
    mw.set_button_text(button, "Registrace")
    mw.set_button_x(button, 20)
    mw.set_button_y(button, 60)
    mw.set_button_click_handler(button, form_button_click_handler)
    return button
```

Všimněme si, že procedura `form_button_click_handler` zatím není definována. Dále vytvoříme konstruktor pro formulář:

```
def make_form(window):
    label = make_form_label(window)
    entry = make_form_entry(window)
    button = make_form_button(window)
    form = ["form", label, entry, button]
    mw.set_button_data(button, form)
    return form
```

Všimněme si, že formulář bude v datech tlačítka. Bude nám stačit jediný selektor:

```
def get_form_entry(form):
    return form[2]
```

Na závěr definujeme příkaz tlačítka formuláře:

```
def form_button_click_handler(button):
    form = mw.get_button_data(button)
    entry = get_form_entry(form)
    text = mw.get_entry_text(entry)
    print("Registrace uživatele:", text)
```

Zbývá otevřít okno s formulářem:

```
window = mw.display_window()
form = make_form(window)
```

Otázky a úkoly na cvičení

1. Vytvořte abstraktní datovou strukturu pro počítadlo nazvanou `counter`. Struktura bude mít jedinou položku `value`, kde hodnota položky je číslo udávající hodnotu počítadla. Napište proceduru `inc_counter`, která zvýší hodnotu počítadla o jedna. Příklad použití:

```
>>> counter = make_counter()
>>> get_counter_value(counter)
0
>>> inc_counter(counter)
>>> get_counter_value(counter)
1
>>> set_counter_value(counter, 0)
>>> get_counter_value(counter)
0
```

2. Napište proceduru `make_change_button`, která vytvoří tlačítko. Při kliku na tlačítko se změní jeho text.

3. Napište proceduru `make_integer_entry`, která vytvoří textové pole, do kterého půjdou zadat jen cifry.

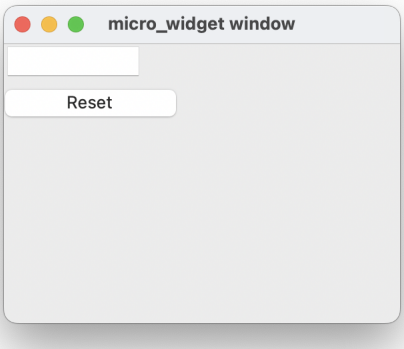
4. Vytvořte abstraktní datovou strukturu `integer_entry`, která bude představovat pole na zadávání celých nezáporných čísel. Struktura bude mít položku `value` udávající celočíselnou hodnotu zadanou uživatelem. Procedura `is_integer_entry_valid` rozhodne, zda uživatel zadal platnou hodnotu. Příklad použití:

   ```
   >>> window = mw.display_window()
   >>> integer_entry = make_integer_entry(window)
   >>> set_integer_entry_value(integer_entry, 20)
   >>> get_integer_entry_value(integer_entry)
   20
   >>> is_integer_entry_valid(integer_entry)
   True
   >>> mw.set_entry_text(integer_entry, "1b2")
   >>> is_integer_entry_valid(integer_entry)
   False
   ```

5. Vytvořte textové pole s tlačítkem pro smazaní. Kód:

   ```
   window = mw.display_window()
   resetable_entry = make_resetable_entry(window)
   ```

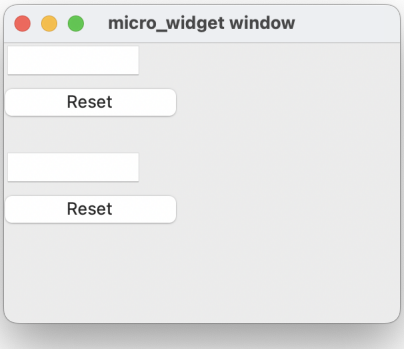
   vytvoří okno:



6. Doplňte do textového pole s tlačítkem pro reset položky pro $x$ovou a $y$ovou souřadnici. Například kód:

```
window = mw.display_window()
resetable_entry1 = make_resetable_entry(window)
resetable_entry2 = make_resetable_entry(window)
set_resetable_entry_y(resetable_entry2, 80)
```

vytvoří dvě pole:



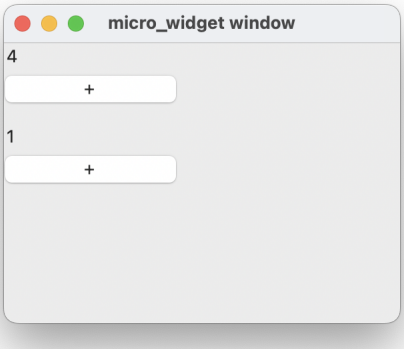
Souřadnice můžeme i číst. Například:

```
>>> get_resetable_entry_y(resetable_entry2)
80
```

7. Rozšiřte počítadlo z prvního úkolu tak, aby se zobrazovalo v okně. Počítadlo se zobrazí jako popisek udávající hodnotu počítadla a tlačítko na jeho inkrementaci. Například:

```
window = mw.display_window()
counter1 = make_counter(window)
counter2 = make_counter(window)
set_counter_y(counter2, 60)
set_counter_value(counter1, 4)
inc_counter(counter2)
```

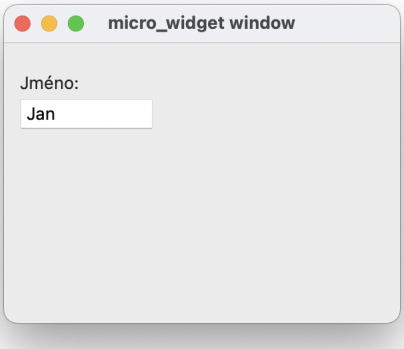
vytvoří okno s dvěma počítadly:



8. Vytvořte abstraktní datovou strukturu `entry_field`, která bude představovat textové pole s popiskem. Například kód:

```
window = mw.display_window()
entry_field = make_entry_field(window, "Jméno:")
set_entry_field_x(entry_field, 10)
set_entry_field_y(entry_field, 20)
set_entry_field_text(entry_field, "Jan")
```

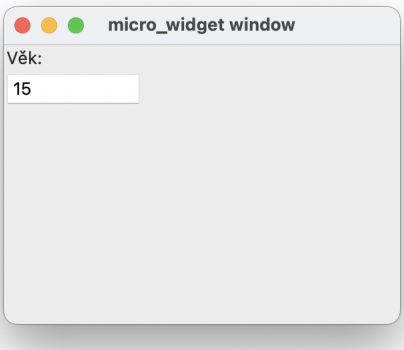
vytvoří okno:



9. Vytvořte číselné pole s popiskem. Pod polem se zobrazí chybová hláška v případě, že uživatel nezadá číslo. Například kód:

```
window = mw.display_window()
integer_field = make_integer_field(window, "Věk:")
set_integer_field_value(integer_field, 15)
```

vytvoří okno:



Pokud uživatel zadá hodnotu, která není číslem, okno uživatele informuje o chybě: