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

9. Funkcionální uživatelské rozhraní

verze z 5. prosince 2023

1 Lexikální uzávěry

Prostředí udává hodnoty proměnných. Například:

`x`	`1`
`y`	`2`

je prostředí, kde hodnota proměnné `x` je číslo jedna a proměnné `y` číslo dva.

Prostředí může mít **rodiče**, kterým je opět prostředí. Jediné prostředí, které nazýváme **globální prostředí**, nemá žádného rodiče.

Vykonávání kódu probíhá vždy v nějakém prostředí, které nazýváme **aktuální**. Při spuštění programu se kód programu vykoná v globálním prostředí. Pokud neřekneme jinak, myslíme vykonáním programu jeho vykonání v globálním prostředí.

Zjišťování hodnoty proměnné ***`variable`*** v prostředí E probíhá následovně:

- Pokud prostředí E udává hodnotu V proměnné ***`variable`***, pak je V výsledkem.
- Pokud prostředí E neudává hodnotu proměnné ***`variable`***, ale má rodiče E_P, pak je výsledek hodnota proměnné ***`variable`*** v prostředí E_P.
- Jinak proměnná ***`variable`*** nemá hodnotu v prostředí E.

Například pokud by prostředí G:

`x`	`1`
`y`	`2`

bylo globální a prostředí E:

`x`	`2`
`z`	`3`

mělo za rodiče prostředí G, pak by v prostředí E měla proměnná `x` hodnotu 2, `y` hodnotu 2, `z` hodnotu 3 a proměnná `a` by hodnotu v prostředí E neměla.

Vyhodnocování výrazů v prostředí probíhá podle obvyklých pravidel. Například vyhodnocením výrazu:

```
x + y
```

povede v prostředí:

x	1
y	2

k hodnotě 3.

Vykonání příkazu:

```
variable = value
```

v prostředí E proběhne tak, že se v prostředí E vytvoří proměnná ***variable*** s hodnotou ***value***. Připomeňme si, že neumožňujeme změnu hodnoty proměnných.

Například po vykonání:

```
y = x + 1
```

v prostředí:

x	2

se prostředí změní na:

x	2
y	3

Každá funkce je určena:

1. parametry,
2. tělem
3. a **prostředím vzniku**.

Příkaz definující funkci:

```
def function(param1, ..., paramn):
    body
```

vytvoří funkci s parametry ***param***1, ..., ***param***n, tělem, ***body***, kde prostředím vzniku bude aktuální prostředí, a v aktuálním prostředí vytvoří proměnnou ***function***, kde hodnotou bude právě vytvořená funkce.

Například program:

```
def f(x, y):
    return x + y
```

přidá do globálního prostředí proměnnou `f`, která za hodnotu bude mít funkci s parametry `x` a `y`, tělem `return x + y` a prostředím vzniku rovným globálnímu prostředí.

Volání funkce probíhá tak, že

1. se nejprve vytvoří prostředí E, kde parametry funkce budou mít postupně hodnoty argumentů. Rodičem prostředí E bude prostředí vzniku funkce.
2. Poté se vykoná tělo funkce v prostředí E.

Například:

```
f(1, 2)
```

způsobí vytvoření prostředí E:

`x`	`1`
`y`	`2`

Rodičem prostředí E je globální prostředí (prostředí vzniku funkce `f`). Poté se v prostředí E vykoná tělo funkce:

```
return x + y
```

To povede na návratovou hodnotu 3.

Definujme si funkci:

```
def g(y):
    def h(x):
        return x + y
    return h
```

Zavoláním:

```
h1 = g(2)
```

vznikne prostředí E_1:

`y`	`2`

v kterém se vykoná:

```
def h(x):
   return x + y
return h
```

Funkce `h` bude mít prostředí vzniku E_1. Zavoláním:

```
h1(3)
```

vznikne prostředí E_2:

x	3

Rodičem E_2 bude prostředí vzniku funkce `h1` tedy prostředí E_1.

V prostředí E_2 se vykoná:

```
return x + y
```

Hodnota proměnné `x` se nalezne v prostředí E_2 a hodnota proměnné `y` v prostředí E_1 (rodič E_2). Výsledkem volání tedy bude 3.

Vyhodnocení lambda výrazu:

`lambda` *param*1`, ...,` *param*n`:` *body*

v prostředí E získáme funkci s parametry *param*1, ..., *param*n, tělem `return` *body* a prostředím vzniku E.

2 Funkcionální uživatelské rozhraní

Budeme potřebovat knihovnu Functional Micro Widget (`fmw`) verze 2. Manuál ke knihovně se nalézá v souboru `09_fmw.py`.

Prvky uživatelského rozhraní definujeme pomocí příslušných funkcí z knihovny. Například popisek s textem `"Ahoj"`:

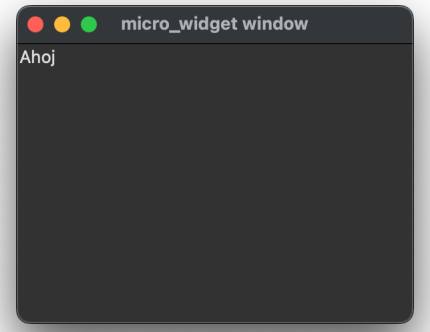
```
>>> label("Ahoj")
label('Ahoj', 0, 0)
```

Funkce `display_window` zavolaná na prvek uživatelského rozhraní zobrazí okno se zadaným prvkem.

Například příkaz:

```
>>> display_window(label("Ahoj"))
```

zobrazí okno:



Souřadnice prvku můžeme zadat přímo:

```
>>> label("Ahoj", 10, 50)
label('Ahoj', 10, 50)
```

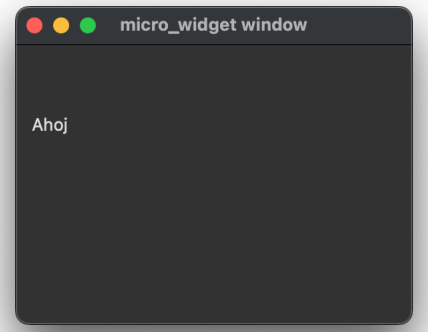
nebo posunutím:

```
>>> moved(label("Ahoj"), 10, 50)
label('Ahoj', 10, 50)
```

V obou případech dostaneme stejný prvek, který můžeme dát do okna:

```
>>> display_window(moved(label("Ahoj"), 10, 50))
```

uvidíme:



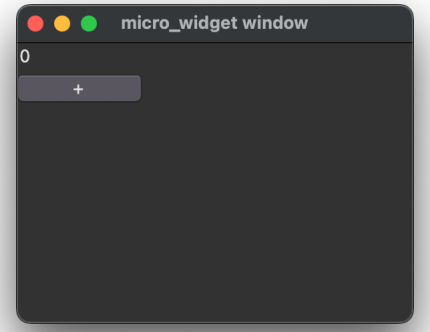
Složený prvek zadáme funkcí `group` očekávající dva prvky, které chceme složit:

```
>>> group(label("0"), moved(button("+"), 0, 20))
group(label('0', 0, 0), button('+', None, 0, 20))
```

Příkaz:

```
>>> display_window(group(label("0"), moved(button("+"), 0, 20)))
```

zobrazí okno:



Obsah okna můžeme určit funkcí:

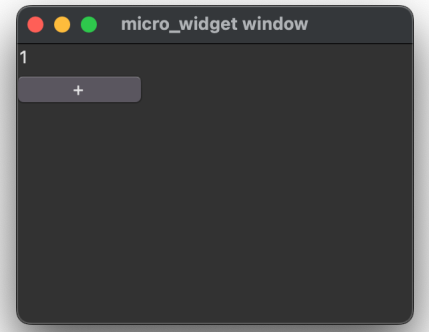
```
def counter(value):
    return group(label(str(value)),
                 moved(button("+"), 0, 20))
```

Obsah předchozího okna můžeme zadat i takto:

```
>>> counter(0)
group(label('0', 0, 0), button('+', None, 0, 20))
```

Zadáme jinou hodnotu počítadla:

```
>>> display_window(counter(1))
```



Funkci `display_window` můžeme zavolat i následovně.

```
display_window(content, initial_state)
```

Argument *content* musí být funkce jednoho parametru, která vrací ovládací prvek, a argument *initial_state* je libovolná hodnota. Nejprve se zavolá *content* na *initial_state*. Tím se obdrží ovládací prvek, který bude obsahem zobrazeného okna.

Naposledy zobrazené okno získáme i příkazem:

```
>>> display_window(counter, 1)
```

Zobrazené okno má **stav**, kterým je na začátku hodnota *initial_state*. Druhý argument tlačítka udává stav, do kterého okno přejde při jeho stisku:

```
button(text, next_state)
```

Upravíme definici funkce `counter` tak, aby další stav po stisku tlačítka byl o jedna větší než aktuální stav:

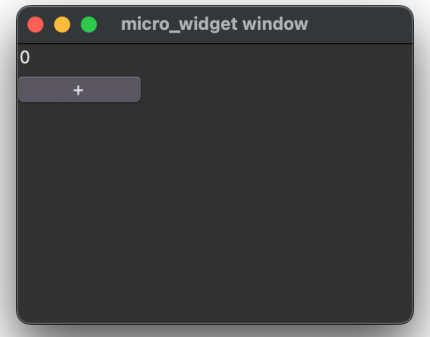
```
def counter(value):
    return group(label(str(value)),
                 moved(button("+", value + 1), 0, 20))
```

Pokud by například byl aktuální stav nula, pak další stav je číslo jedna:

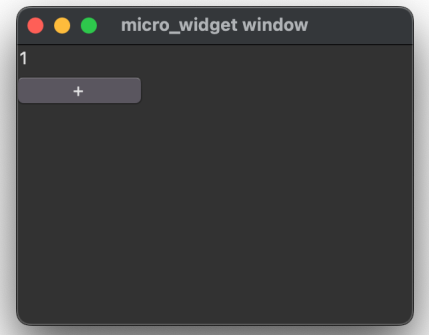
```
>>> counter(0)
group(label('0', 0, 0), button('+', 1, 0, 20))
```

Zobrazíme počítadlo:

```
>>> display_window(counter, 0)
```



Po kliku na tlačítko se aktuální stav okna změní na číslo jedna a v důsledku toho se hodnota počítadla zvětší:



Poslední varianta volání funkce `display_window` je následující.

```
display_window(content, initial_state, update)
```

Argument ***update*** musí být funkce dvou parametrů. Funkce se volá v případě, že dojde k **vyvolání akce**. V takové situacise funkce ***update*** zavolá na aktuální stav a vyvolanou akci a vrátí následující stav okna.

Druhý argument tlačítka se také nazývá **akce**. Při stisku tlačítka se vyvolá jeho akce.

Opět upravíme funkci `counter`:

```
def counter(value):
    return group(label(str(value)),
                 moved(button("+", 1), 0, 20))
```

Všimněte si, že nyní je akce tlačítka vždy číslo jedna:

```
>>> counter(5)
group(label('5', 0, 0), button('+', 1, 0, 20))
```

Přidáme funkci na zpracování vyvolaných akcí:

```
def update_counter_value(value, increment):
    return value + increment
```

Pokud by aktuální stav bylo číslo pět a akce číslo jedna, pak funkce určí následující stav:

```
>>> update_counter_value(5, 1)
6
```

Počítadlo zobrazíme příkazem:

```
display_window(counter, 0, update_counter_value)
```

Druhý argument textového pole je akce:

```
entry(text, action)
```

Při změně textu v poli se vyvolá akce:

```
[action, text]
```

kde *text* je změněný text.

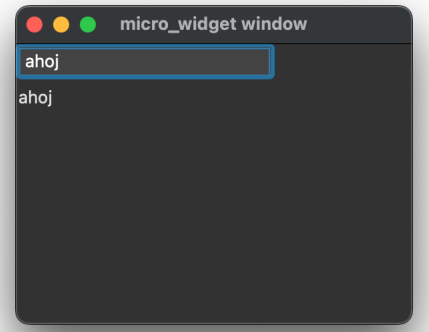
Například tento jednoduchý příklad zopakuje pod textovým polem v něm zadaný text:

```
def content(string):
    return group(entry(string, True),
                 moved(label(string), 0, 30))


def update(string, action):
    return action[1]

display_window(content, "", update)
```

Po napsání `ahoj` do textového pole obdržíme:



Otázky a úkoly na cvičení

1. Uvažujme program:

```
def succ(x):
    return x + 1

def square(x):
    return x ** 2

def comp(f, g):
    return lambda x: f(g(x))
```

Zakreslete všechna prostředí a funkce, které vzniknou vykonáním programu a vyhodnocením:

```
>>> comp(succ, square)(3)
10
```