
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

2. Objekty

verze z 2. října 2024

1 Opakování

Připomeňme si definici abstraktní datové struktury bodu:

```
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]

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

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

Podívejme se na příklad použití:

```
>>> point = make_point(3, 4)
>>> get_point_x(point)
3
>>> set_point_x(point, 5)
>>> get_point_x(point)
5
```

V procedurálním programování jsou složená data uchovávána v podobě datových struktur a kód je udržován v procedurách. Pro tento styl programování je příznačné, že data a kód, který s nimi pracuje jsou oddělené.

2 Objekty

Nyní se podíváme na objektový styl programování, který se liší od procedurálního stylu v tom, že data a kód, který s nimi pracuje, chápeme jako jeden celek nazývaný **objekt**. Například si představme, že máme objekt `label` reprezentující popisek. Jak následující úryvek kódu ukazuje, objekty jsou, podobně jako čísla nebo pole, hodnoty.

```
>>> label
<pomw.Label object at 0x11065fe50>
```

Formát v jakém se objekty tisknou rozebereme za chvíli.

S objektem komunikujeme výhradně pomocí mechanizmu **zasílání zpráv**. Zasláním zprávy říkáme objektu, co chceme, aby udělal, ale nestaráme se o to, jak to má provést. Například zasláním zprávy `set_text` chceme, aby objekt změnil svůj text.

Obecně objektu *`object`* můžeme zaslat zprávu *`message`* s argumenty *`arg1`*, *`arg2`*, ... Argumenty zprávy jsou hodnoty. Mohou to tedy být i objekty. Objekt, kterému zprávu zasíláme, se nazývá **příjemce zprávy**. Zprávu objekt přijme tak, že spustí kód, který má určený pro obsluhu zaslané zprávy. Například zaslání zprávy `set_text` popisku `label` s argumentem `"Pomeranč"` způsobí vykonání kódu, který změní text popisku. Spuštěný kód také určí **návratovou hodnotu**. Například zasláním zprávy `get_text` popisku `label` bez argumentů, vykoná kód, který vrátí text popisku; v našem případě řetězec `"Pomeranč"`.

Objekty mají **vnitřní stav**, kde si mohou uchovávat další hodnoty a tedy i objekty. Například text popisku by byl součástí jeho vnitřního stavu. Změnu vnitřního stavu provedeme zasláním vhodné zprávy. Jak již víme, zaslání zprávy `set_text` popisku způsobí nastavení jeho textu a tedy změnu vnitřního stavu.

Následující výraz objektu *`object`* zašle zprávu *`message`* s argumenty *`arg1`*, *`arg2`*, ...

```
object.message(arg1, arg2, ...) => value
```

Například:

```
>>> label.set_text("Pomeranč")
<pomw.Label object at 0x11065fe50>
>>> label.get_text()
'Pomeranč'
```

Tedy výraz `label.set_text("Pomeranč")` zašle popisku `label` zprávu `set_text` s argumentem `"Pomeranč"`. Návratovou hodnotou je popisek, kterému byla zpráva

zaslána, a výraz `label.get_text()` zašle popisku `label` zprávu `get_text` bez argumentů. Řetězec `"Pomeranč"` je návratovou hodnotou zaslané zprávy.

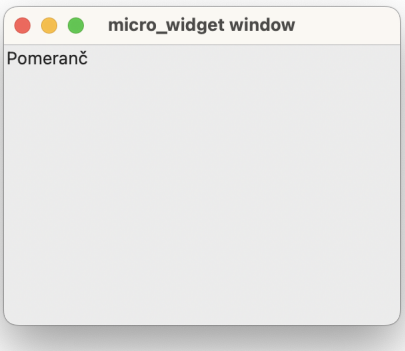
Objekt může přijmout jen některé zprávy. Například objekt `label` může přijmout zprávu `get_text`, ale už ne zprávu `get_color`:

```
>>> label.get_text()
'Pomeranč'
>>> label.get_color()
AttributeError: 'Label' object has no attribute 'get_color'
```

Říkáme, že objekt **rozumí** zprávě, pokud ji přijme. Tedy například objekt `label` rozumí zprávě `get_text` ale nerozumí zprávě `get_color`. Množinu zpráv, kterým objekt rozumí, nazýváme **rozhraní** objektu. Například rozhraní objektu `label` by mohla být množina

$$\{\texttt{get\_text}, \texttt{set\_text}, \texttt{get\_x}, \texttt{set\_x}, \texttt{get\_y}, \texttt{set\_y}, \texttt{move}\}.$$

Vezměme popisek `label`:



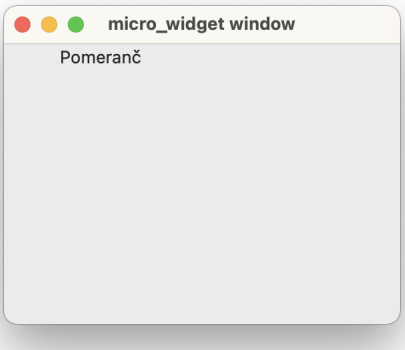
Jeho $x$ová souřadnice je nulová:

```
>>> label.get_x()
0
```

Změna $x$ové souřadnice:

```
>>> label.set_x(40)
<pomw.Label object at 0x11065fe50>
```

se projeví v okně:



Ověříme změnu souřadnice:

```
>>> label.get_x()
40
```

Objekty mají **vlastnosti**, které chápeme jako pojmenované položky. Například popisek má mimo jiné vlastnosti `text`, `x` a `y`. Hodnotu vlastnosti *`property`* objektu získáme zasláním zprávy:

```
object.get_property() => value
```

Například text popisku získáme:

```
label.get_text() => string
```

Nastavení hodnoty vlastnosti provedeme zasláním zprávy:

```
object.set_property(value) => object
```

Například změna textu popisku:

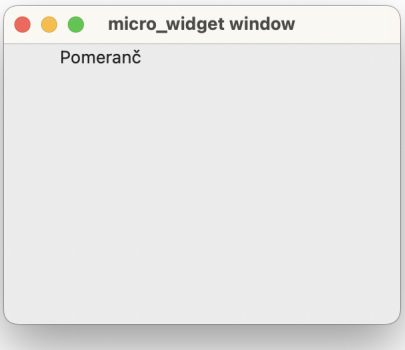
```
label.set_text(string) => label
```

Některé vlastnosti jsou pouze pro čtení. Objekt pak nerozumí zprávě pro nastavení hodnoty vlastnosti.

Posun popisku provedeme zasláním zprávy `move`, kde jako argumenty zadáme přírůstky obou souřadnic:

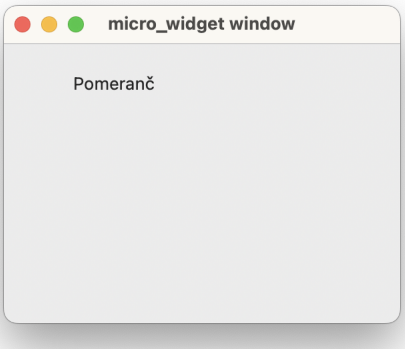
```
label.move(dx, dy) => label
```

Například popisek:



posuneme o deset a dvacet pixelů v $x$ové a $y$ové souřadnici zasláním zprávy:

```
>>> label.move(10, 20)
<pomw.Label object at 0x11065fe50>
```



Pokud je to možné, zaslání zprávy vrací jako hodnotu příjemce zprávy. Například nastavení hodnoty vlastnosti zprávou `set_propery` nebo posun zprávou `move`. Tím umožníme, aby objektu bylo možné zaslat více zpráv v jednom výrazu. Tato technika se nazývá **zřetězování** zasílání zpráv.

Například:

```
>>> label.set_text("Rajče").move(-50, -20)
<pomw.Label object at 0x11065fe50>
```

Zde se nejprve popisku zaslala zpráva `set_text` a poté zpráva `move`. Tedy výraz je ekvivalentní s:

```
(label.set_text("Rajče")).move(-50, -20)
```

3 Třídy

Strukturu a chování objektů určují **třídy**. Třída objektu tedy určuje jeho rozhraní. Jména tříd začínají velkým písmenem. Například třída pro popisek má jméno `Label`. Víceslovné názvy spojujeme velbloudí notací. Například `ColoredLabel`.

Každý objekt je **přímou instancí** určité třídy. Například objekt `label` je přímou instancí třídy `Label`.

Tisk objektu začíná jménem třídy, jejíž je objekt přímou instancí. Například:

```
>>> label
<pomw.Label object at 0x11065fe50>
```

Kromě třídy je při tisku uveden i jednoznačný **identifikátor** objektu. Zde číslo `0x11065fe50` zapsané v šestnáctkové soustavě.

Novou přímou instanci *`object`* třídy *`Class`* vytvoříme výrazem:

```
Class() => object
```

Například:

```
>>> label2 = Label()
>>> label2
<pomw.Label object at 0x110490ad0>
>>> label
<pomw.Label object at 0x11065fe50>
```

Všimněte si, že různé přímé instance téže třídy se liší identifikátorem.

Vestavěný predikát `isinstance` rozhoduje, zda je hodnota objektem, jenž je přímou instancí zadané třídy:

```
>>> isinstance(label, Label)
True
>>> isinstance(label2, Label)
True
>>> isinstance(label, Button)
False
>>> isinstance(1, Button)
False
```

4 Uživatelské rozhraní objektově

Knihovna `pomw` je procedurálně objektová knihovna na vytváření uživatelského rozhraní. Knihovna je objektovou nadstavbou knihovny `micro_widget`. Manuál ke knihovně naleznete v souboru `pomw.pdf`. Knihovnu tvoří soubory `micro_widget.py` a `pomw.py`. Její import provedeme příkazem:

```
from pomw import *
```

Opět je knihovnu potřeba používat v prostředí IDLE.

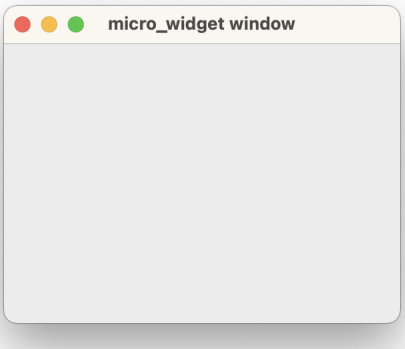
Knihovna nabízí třídy:

- `Window` (okno)
- `Label` (popisek)
- `Button` (tlačítko)
- `Entry` (textové pole)
- `Group` (skupina)

Okna jsou přímé instance třídy `Window`. Vytvoření nové instance zobrazí nové okno.

Například:

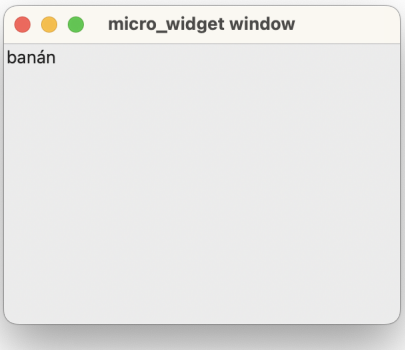
```
>>> window2 = Window()
```



Vlastnost okna `widget` určuje jeho obsah.

Změnou vlastnosti `widget` můžeme například do okna vložit popisek:

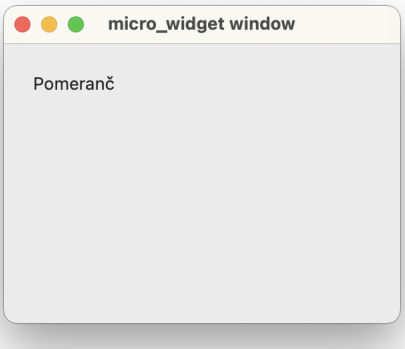
```
>>> label2.set_text("banán")
<pomw.Label object at 0x110490ad0>
>>> window2.set_widget(label2)
<pomw.Window object at 0x1106b3910>
```



Následuje celý jednoduchý ukázkový program.

```
from pomw import *
window = Window()
label = Label().set_text("Pomeranč").move(20, 20)
window.set_widget(label)
```

Po jeho spuštění se zobrazí okno:

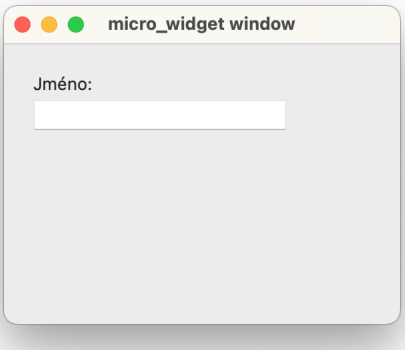


Okno může obsahovat jediný ovládací prvek. Pokud chceme do okna vložit více prvků, musíme z nich nejprve vytvořit skupinu. Skupiny jsou instance třídy `Group`. Prvky skupiny určuje vlastnost `items`.

Například:

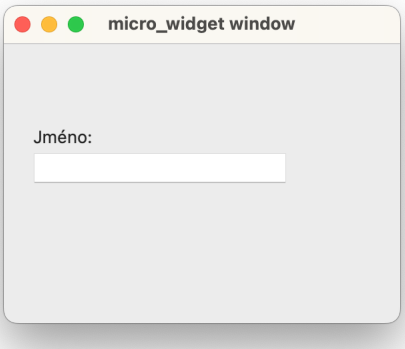
```
group = Group().set_items([label, entry])
window.set_widget(group)
```

Obsah okna se změní:

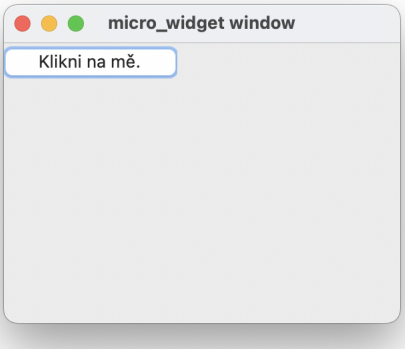


Podobně jako popisek, lze i skupinu posunout zasláním zprávy `move`. Dojde k posunutí všech prvků skupiny:

```
>>> group.move(0, 40)
<pomw.Group object at 0x108f40e10>
```



Reakce na uživatelský vstup se stále provádí pomocí procedur. Například pokud máme tlačítko `button` v okně:



Můžeme nastavením vlastnosti `click_handler` tlačítka určit jaká procedura se má zavolat poté, co uživatel na tlačítko klikne:

```
def button_click_handler(button):
    print("Bylo kliknuto na tlačítko.")

button.set_click_handler(button_click_handler)
```

Kliknutím na tlačítko se vytiskne:

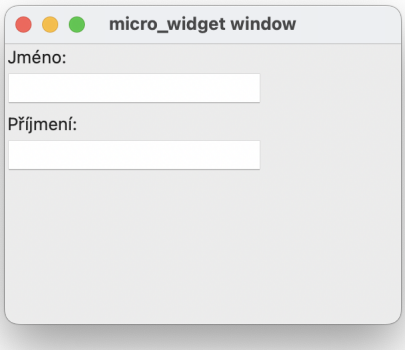
```
Bylo kliknuto na tlačítko.
```

Následuje porovnání knihoven `micro_widget` a `pomw`. Prvky v `pomw` můžeme vytvářet v libovolném pořadí. Například můžeme vytvořit popisek před oknem, do kterého má být umístěn. Knihovna `pomw` má kratší názvy akcí s prvky. Akce

nemusí obsahovat typ prvku, pro který jsou určeny. Například `set_text` oproti `set_label_text`. Prvky v `pomw` nemají destruktor. O jejich odstranění se stará knihovna při změnách vlastností `widget` a `items`. V knihovně `pomw` můžeme shlukovat prvky do skupin. Knihovna `pomw` stále reaguje na uživatelský vstup pomocí procedur.

Otázky a úkoly na cvičení

1. Vytvořte okno s obsahem:

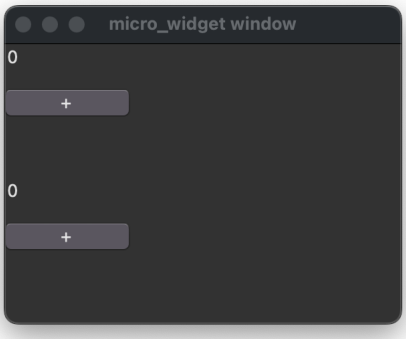


   Na opakující části kódu si vytvořte vhodné procedury.

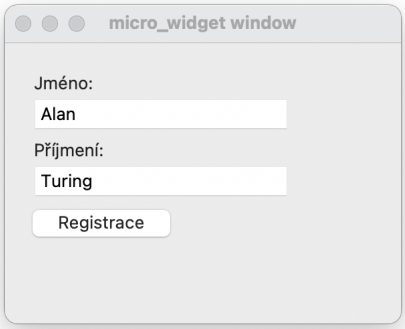
2. Vytvořte počítadlo jako ovládací prvek. Počítadlo bude zobrazovat svou hodnotu a klikem na tlačítko hodnotu zvýší o jedna. Napište si proceduru `make_counter` na vytvoření počítadla.Například kód:

```
window = Window()
counter1 = make_counter()
counter2 = make_counter().move(0, 100)
group = Group().set_items([counter1, counter2])
window.set_widget(group)
```

   vytvoří okno s dvěma počítadly:



3. Vytvořte jednoduchý registrační formulář:



Klik na tlačítko způsobí tisk zadaných údajů:

```
['Alan', 'Turing']
```

4. Vytvořte ovládací prvek na zadání hesla. Heslo bude uživatel zadávat dvakrát. Prvek bude uživatele po každé změně jednoho z textových polí informovat, zda jsou hesla stejná. Například: