Úvod do programovacích stylů

Přednáška 1. Procedurální programování

verze z 20. září 2023

Jan Laštovička

KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI

Obsah

Úvod

Úvod

Cíl kursu:

- poznat různé programovací styly
- řešit problém vhodným stylem
- zlepšit čitelnost kódu

Úvod

Cíl kursu:

- poznat různé programovací styly
- řešit problém vhodným stylem
- zlepšit čitelnost kódu

Budeme používat:

- programovací jazyk Python
- vývojové prostředí IDLE

Obsah

Procedury

Procedury

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

Procedury

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

Procedura:

- parametry
- tělo

Procedury

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

Procedura:

- parametry
- tělo

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

Procedury

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

Procedura:

- parametry
- tělo

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

Volání procedury:

- argumenty
- návratová hodnota

Procedura jako hodnota

Procedura jako hodnota

Jméno není součástí procedury.

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

Procedura jako hodnota

Jméno není součástí procedury.

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

Procedura je hodnota.

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

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

Obsah

Datové struktury

Datové struktury

- pole
- prvky pole jsou položky

Datové struktury

- pole
- prvky pole jsou položky

Bod: `[x, y]`

Datové struktury

- pole
- prvky pole jsou položky

Bod: `[x, y]`

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

Datové struktury

- pole
- prvky pole jsou položky

Bod: `[x, y]`

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

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

Datové struktury

- pole
- prvky pole jsou položky

Bod: `[x, y]`

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

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

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

Konstruktor a selektory

Konstruktor a selektory

Konstruktor

- procedura vytvářející datovou strukturu

Konstruktor a selektory

Konstruktor

- procedura vytvářející datovou strukturu

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

Konstruktor a selektory

Konstruktor

- procedura vytvářející datovou strukturu

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

Selektory:

- vrací položky datové struktury

Konstruktor a selektory

Konstruktor

- procedura vytvářející datovou strukturu

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

Selektory:

- vrací položky datové struktury

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

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

Konstruktor a selektory

Konstruktor a selektory

Ukázka:

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

Konstruktor a selektory

Ukázka:

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

Stará verze:

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

Konstruktor a selektory

Ukázka:

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

Stará verze:

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

Použití:

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

Obsah

Abstraktní datové struktury

- dány pouze konstruktorem a selektory

Abstraktní datové struktury

- dány pouze konstruktorem a selektory

Bod:

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

Abstraktní datové struktury

- dány pouze konstruktorem a selektory

Bod:

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

Výhody:

- čitelný kód
- snadná změna reprezentace datové struktury

Změna reprezentace bodu

Změna reprezentace bodu

`[`x`, `y`]` $\rightarrow$ `["point", `x`, `y`]`

Změna reprezentace bodu

`[`x`, `y`]` $\rightarrow$ `["point", `x`, `y`]`

Potřeba změnit jen 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]
```

Změna reprezentace bodu

Změna reprezentace bodu

Ostatní kód není potřeba měnit.

Změna reprezentace bodu

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)
```

Změna reprezentace bodu

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]
```

Typové predikáty

Typové predikáty

rozhodují, zda je hodnota určitou datovou strukturou

Typové predikáty

rozhodují, zda je hodnota určitou datovou strukturou

Typový predikát bodu:

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

Typové predikáty

rozhodují, zda je hodnota určitou datovou strukturou

Typový predikát bodu:

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

Test:

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

Nedestruktivní procedury

Nedestruktivní procedury

- nemění své argumenty
- konstruktory a selektory nejsou destruktivní
- `move_point` není destruktivní

Nedestruktivní procedury

- nemění své argumenty
- konstruktory a selektory nejsou destruktivní
- `move_point` není destruktivní

Procedura `move_point` 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]
```

Obsah

Mutátory

Mutátory

- mění položky datové struktury
- patří k definici abstraktní datové struktury
- jsou destruktivní

Mutátory

- mění položky datové struktury
- patří k definici abstraktní datové struktury
- jsou destruktivní

Mutátory bodu:

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

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

Příklad

Příklad

```
>>> 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]
```

Destruktivní posun bodu

Destruktivní posun bodu

Definice:

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

Destruktivní posun bodu

Definice:

```
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]
```

Výchozí hodnoty položek

Výchozí hodnoty položek

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

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

Výchozí hodnoty položek

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]
```

Obsah

Knihovna `micro_widget`

Knihovna `micro_widget`

Malá knihovna na tvorbu grafického uživatelského rozhraní.

Knihovna `micro_widget`

Malá knihovna na tvorbu grafického uživatelského rozhraní.

- procedurální knihovna

Knihovna `micro_widget`

Malá knihovna na tvorbu grafického uživatelského rozhraní.

- procedurální knihovna

Prvky:

- okno (`window`)
- popisek (`label`)
- tlačítko (`button`)
- textové pole (`entry`)

Knihovna `micro_widget`

Malá knihovna na tvorbu grafického uživatelského rozhraní.

- procedurální knihovna

Prvky:

- okno (`window`)
- popisek (`label`)
- tlačítko (`button`)
- textové pole (`entry`)

Prvky mají destruktor.