Algoritmy a programování

Objekty

Vojtěch Vonásek

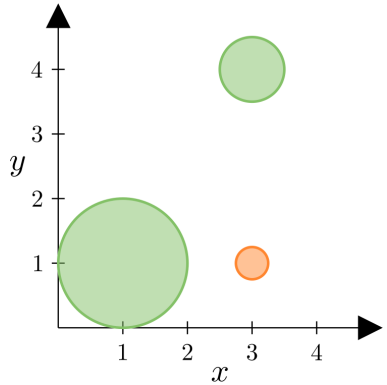
Department of Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague

Data s více atributy: motivační příklad

- Máme $n = 3$ kružnice, známe polohu (x, y), rádius r a barvu c
- Chceme najít největší červenou
- Jak reprezentovat taková data?

Data s více atributy: motivační příklad

- Máme $n = 3$ kružnice, známe polohu (x, y), radius r a barvu c

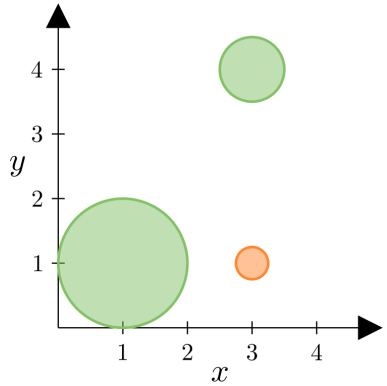


Co atribut, to proměnná

- Na každý atribut každé kružnice použijeme jednu proměnnou
- `x1,y1,r1,c1, ... x3,y3,r3,c3`
- Přidání dalších kružnic nebo atributů zvyšujem počet proměnných
- Funkce musí používat hodně argumentů
- Zvyšuje množství chyb, těžko se ladí, těžko se udržuje
- **Nevhodné, nepoužíváme**

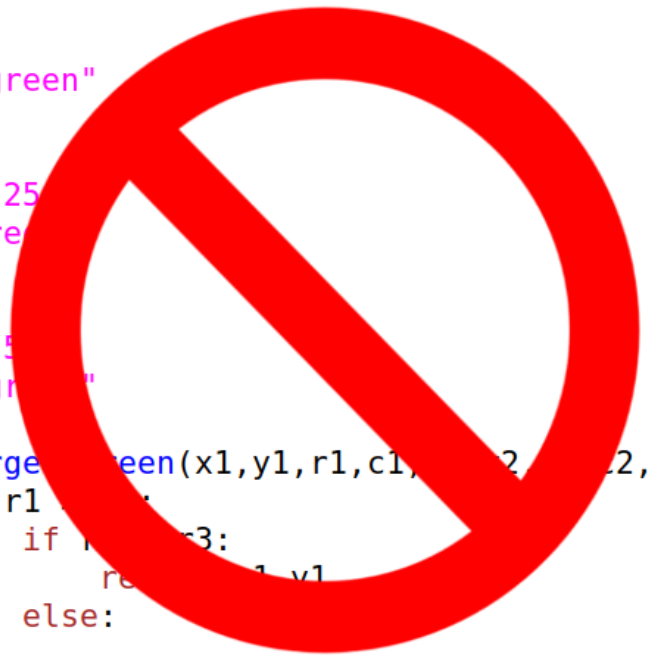
Data s více atributy: motivační příklad

- Máme $n = 3$ kružnice, známe polohu (x, y), radius r a barvu c



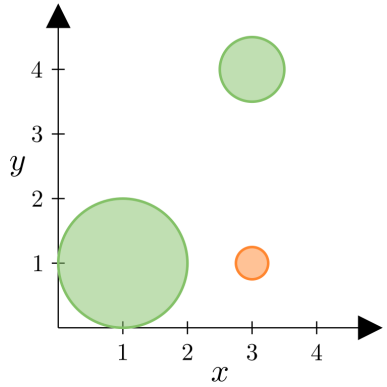
Co atribut, to proměnná

```
 1 x1 = 1
 2 y1 = 1
 3 r1 = 1
 4 c1 = "green"
 5 x2 = 3
 6 y2 = 1
 7 r2 = 0.25
 8 c2 = "re
 9 x3 = 3
10 y3 = 4
11 r3 = 0.5
12 c3 = "gr
13
14 def large  een(x1,y1,r1,c1,  2,  2,x1,y3,c3,r3):
15     if r1
16         if  3:
17             re  1 y1
18         else:
19             ...
20
21 x,y = largestGreen(x1,y1,r1,c1,x2,y2,r2,c2,x1,y3,c3,r3)
```



Data s více atributy: motivační příklad

- Máme $n = 3$ kružnice, známe polohu (x, y), radius r a barvu



Atributy jsou uloženy v poli (nebo tuple)

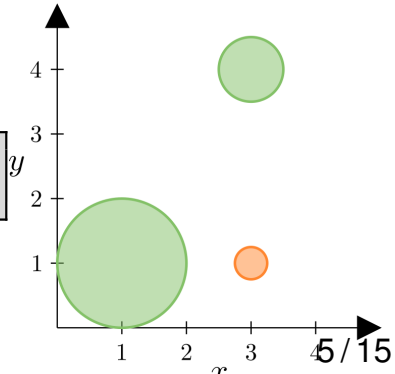
- Kružnici reprezentujeme polem `c = [x,y,radius,color]`
- Vhodné pro malé počty atributů
- ✓ Snižuje počet argumentů funkcí, počet globálních proměnných
- ✗ Položky nejsou pojmenované ⇒ musíme si pamatovat pořadí

```
1 c1 = [3,1,0.25, "red"]
2 c2 = [1,1,1, "green"]
3 c3 = [3,4,0.5, "green"]
4 circles = [c1,c2,c3]
5
6 print("Color␣of␣circle[2]", circles[0][3])
```

Data s více atributy: motivační příklad

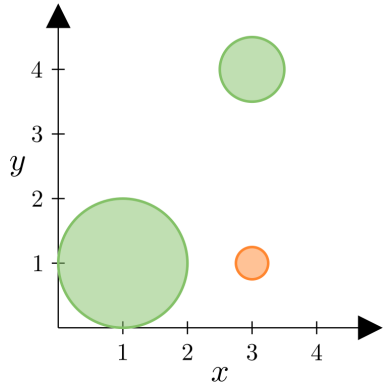

```
1  def findLargest(circles):
2      #circles = [ c1, c2, ... ]
3      #c_i = [ x_i, y_i, radius_i, color_i ]
4      maxCircle = None
5      for c in circles:
6          x,y,rad,color = c
7          if color == "green" and (maxCircle == None or rad >
               maxCircle[2]):
8              maxCircle = c
9      return maxCircle
10
11 c1 = [3,1,0.25, "red"]
12 c2 = [1,1,1, "green"]
13 c3 = [3,4,0.5, "green"]
14 circles = [c1,c2,c3]
15
16 largest = findLargest(circles)
17 print("Largest␣green␣circle:", largest)
```

```
Largest green circle: [1, 1, 1, 'green']
```



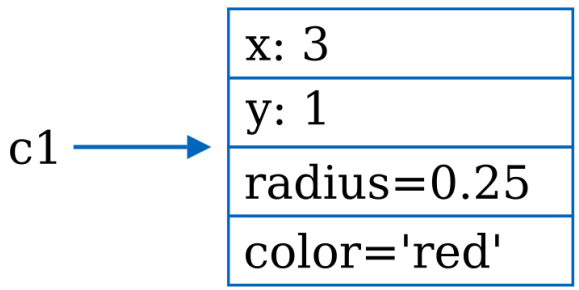
Data s více atributy: motivační příklad

- Máme $n = 3$ kružnice, známe polohu (x, y), radius r a barvu



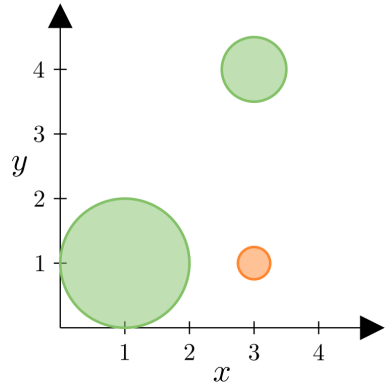
Záznam (record)

- Abstraktní datová struktura obsahující více položek
- Položky jsou pojmenované, mohou mít různé typy
- Python nemá datový typ Record, ale lze použít Object



Data s více atributy: motivační příklad

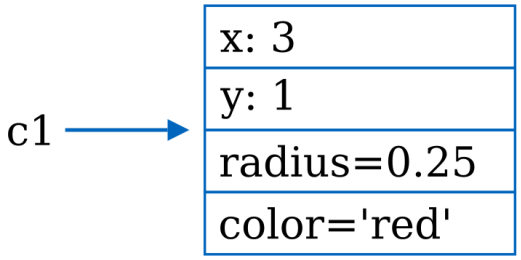
- Máme $n = 3$ kružnice, známe polohu (x, y), radius r a barvu



Záznam (record)

- Abstraktní datová struktura obsahující více položek
- Položky jsou pojmenované, mohou mít různé typy
- Python nemá datový typ Record, ale lze použít Object

```
1  class Circle:
2      def __init__(self,x,y,radius,color):
3          self.x = x
4          self.y = y
5          self.radius = radius
6          self.color = color
7
8  c1 = Circle(1,1,1,"green")
9  c2 = Circle(3,1,0.25,"red")
10 c3 = Circle(3,4,0.5,"green")
11
12 print("radius c1:", c1.radius)
13 print("color c2:", c2.color)
```




```
radius c1: 1
color c2: red
```

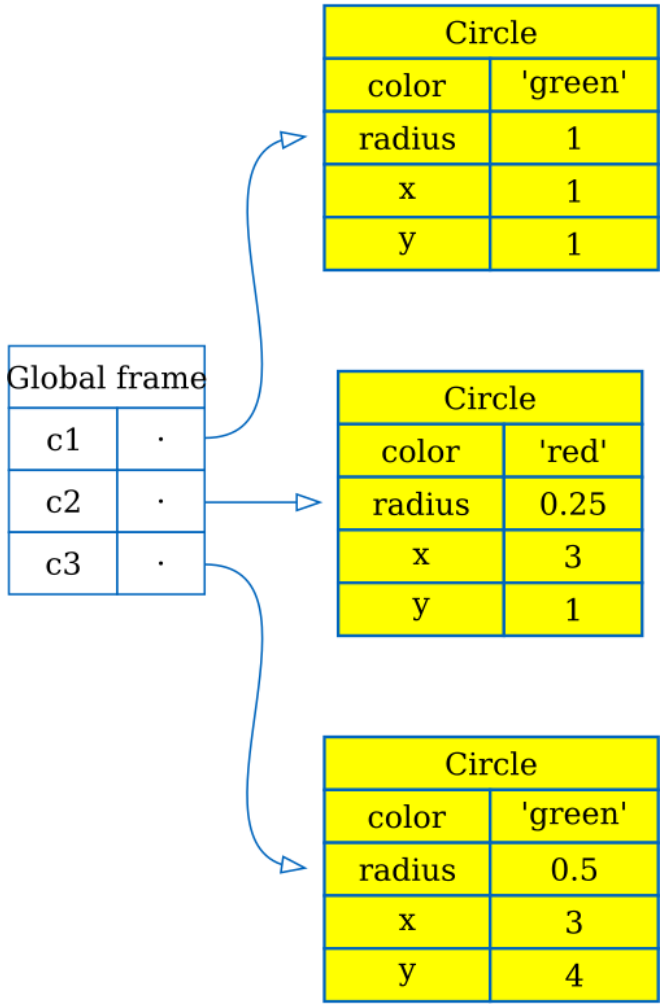
Data s více atributy: motivační příklad


```
1  class Circle:
2      def __init__(self,x,y,radius,color):
3          self.x = x
4          self.y = y
5          self.radius = radius
6          self.color = color
7
8  c1 = Circle(1,1,1,"green")
9  c2 = Circle(3,1,0.25,"red")
10 c3 = Circle(3,4,0.5,"green")
11
12 print("radius␣c1:", c1.radius)
13 print("color␣c2:", c2.color)
```

- `__init__()` je konstruktor, volá se při vytvoření objektu
- Vnitřní proměnné objektu se definují jako `self.jmenoPromenne`
- Funkce ve třídě nazýváme metody
- Všechny metody mají jako první argument `self`

Data s více atributy: motivační příklad


```
1  class Circle:
2      def __init__(self,x,y,radius,color):
3          self.x = x
4          self.y = y
5          self.radius = radius
6          self.color = color
7
8  c1 = Circle(1,1,1,"green")
9  c2 = Circle(3,1,0.25,"red")
10 c3 = Circle(3,4,0.5,"green")
11
12 print("radius␣c1:", c1.radius)
13 print("color␣c2:", c2.color)
```



- Instanci třídy (objekt) vytvoříme jako `variable = className()`
- Například `c1 = Circle(1,1,1,"red")`
- Proměnná `c1` je reference na objekt

Objekty

- Třída je složená datová struktura obsahující
  - data (proměnné)
  - funkce (metody)
- Objekt je instance (proměnná) nějaké třídy
- V Pythonu je instance referencí na objekt

Metody

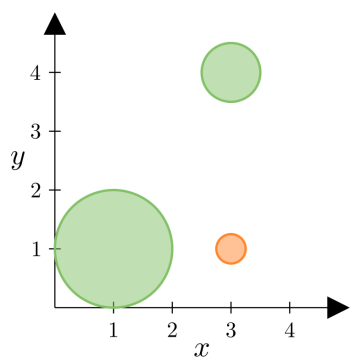
- Metody jsou funkce definované uvnitř třídy
- První argument je vždy reference na objekt
  - Python používá jméno `self`, ale lze použít i jiné
  - Je doporučené používat `self`, zvyšuje to čitelnost programu

```
1 class A:
2     def __init__(self,b):
3         self.b = b
4     def myMethod(self):
5         print("My method, b=", self.b)
6
7 instanceOfA = A(4)
8 instanceOfA.myMethod()
```

Objekt: metody


```
1  class Circle:
2      def __init__(self,x,y,radius,color):
3          self.x = x
4          self.y = y
5          self.radius = radius
6          self.color = color
7      def area(self):
8          return 3.14159*self.radius**2
9      def isIn(self,x,y):
10         return (x-self.x)**2 + (y-self.y)**2 <= self.radius**2
11
12 c1 = Circle(1,1,1,"green")
13 print("Area␣", c1.area())
14 points = [ [0,0], [1,1], [2,1], [3,2] ]
15 for p in points:
16     x,y = p
17     print("Is␣point␣",p, "inside?", c1.isIn(x,y) )
```

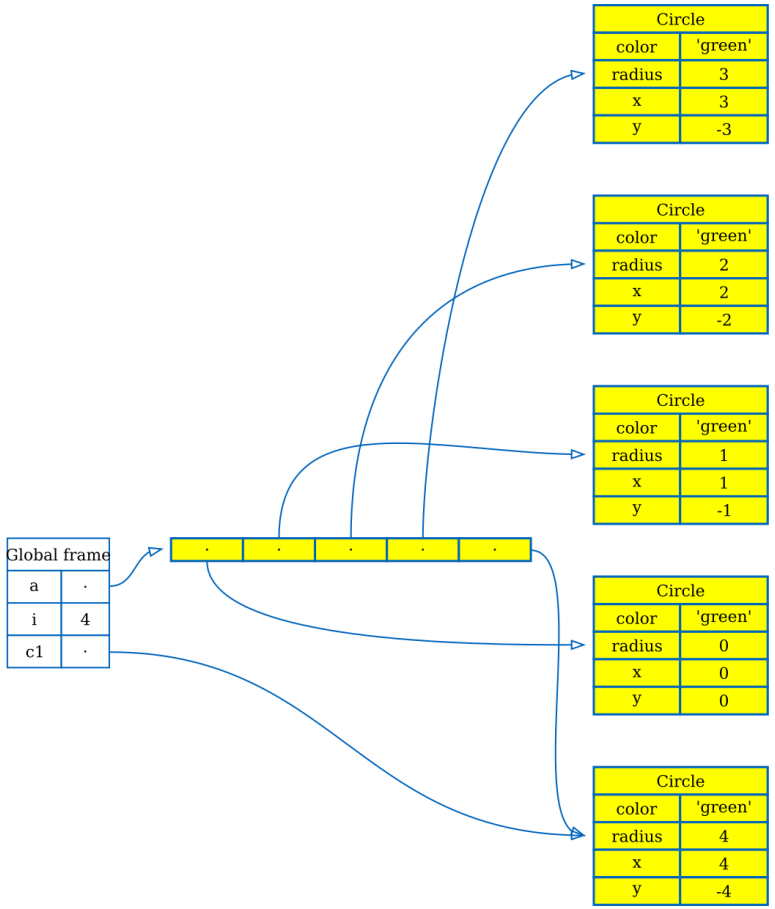
```
Area  3.14159
Is point  [0, 0] inside? False
Is point  [1, 1] inside? True
Is point  [2, 1] inside? True
Is point  [3, 2] inside? False
```



Objekt: reference


```
1  class Circle:
2      def __init__(self,x,y,radius,color):
3          self.x = x
4          self.y = y
5          self.radius = radius
6          self.color = color
7
8  a = []
9  for i in range(5):
10     c1 = Circle(i,-i,i,"green")
11     a.append(c1)
12
13 print("Radius a[1]=", a[1].radius)
14 print("a[-1].x", a[-1].x)
```

```
Radius a[1]= 1
a[-1].x 4
```



Objekt: volání metod

- Z objektu: `self.methodName()`
- Z venku: `objectName.methodName()`

```
1  class Circle:
2      def __init__(self,x,y,radius,color):
3          self.x = x
4          self.y = y
5          self.radius = radius
6          self.color = color
7      def area(self):
8          return 3.14159*self.radius**2
9      def sumOfAreas(self, otherCircle):
10         return self.area() + otherCircle.area()
11
12 c1 = Circle(1,1,1,"green")
13 c2 = Circle(5,5,1,"green")
14 print( c1.sumOfAreas(c2) )
```

```
6.28318
```

Objekt: volání metod


```
1  class Circle:
2      def __init__(self,x,y,radius,color):
3          self.x = x
4          self.y = y
5          self.radius = radius
6          self.color = color
7      def intersect(self, otherCircle):
8          dx = self.x - otherCircle.x
9          dy = self.y - otherCircle.y
10         return (dx**2 + dy**2)**(0.5) <= self.radius + otherCircle.
               radius
11
12 c1 = Circle(0,0,1,"green")
13 c2 = Circle(5,0,2,"green")
14 c3 = Circle(2,0,1.5, "red")
15 print("1 vs 2", c1.intersect(c2))
16 print("1 vs 3", c1.intersect(c3))
17 print("2 vs 3", c2.intersect(c3))
18 print("3 vs 2", c3.intersect(c2))
```

```
1 vs 2 False
1 vs 3 True
2 vs 3 True
3 vs 2 True
```

Sdílení dat mezi objekty


```
1  class State:
2      def __init__(self,data,name):
3          self.data = data
4          self.name = name
5
6      def changeData(self,index, newValue):
7          self.data[ index ] = newValue
8
9      def __repr__(self):
10         return "name:" + str(self.name) + ",␣data:␣" + str(self.
               data)
11
12 a = []
13 mainData = [1,2,3]
14 for i in range(3):
15     a.append( State(mainData, i) )
16
17 print(a)  #call __repr__ on each item in a
18 a[0].changeData(2,"****")
19 print(a)
```

```
[name:0, data: [1, 2, 3], name:1, data: [1, 2, 3], name:2, data:
    [1, 2, 3]]
[name:0, data: [1, 2, '****'], name:1, data: [1, 2, '****'], name
    :2, data: [1, 2, '****']]
```

Sdílení dat mezi objekty


```
1  import copy
2  class State:
3      def __init__(self,data,name):
4          self.data = copy.deepcopy(data)
5          self.name = name
6
7      def changeData(self,index, newValue):
8          self.data[ index ] = newValue
9
10     def __repr__(self):
11         return "name:" + str(self.name) + ",␣data:␣" + str(self.
               data)
12
13 a = []
14 mainData = [1,2,3]
15 for i in range(3):
16     a.append( State(mainData, i) )
17
18 print(a)  #call __repr__ on each item in a
19 a[0].changeData(2,"****")
20 print(a)
```

```
[name:0, data: [1, 2, 3], name:1, data: [1, 2, 3], name:2, data:
    [1, 2, 3]]
[name:0, data: [1, 2, '****'], name:1, data: [1, 2, 3], name:2,
    data: [1, 2, 3]]
```