

B4B33RPH: Řešení problémů a hry

# Python – základní kameny až skály II

Pojmenované argumenty, slovníky, skládání, dědičnost

Tomáš Svoboda, Petr Pošík, **Petr Štibinger**

stibipet@fel.cvut.cz

18.10.2022



Katedra kybernetiky  
Fakulta elektrotechnická  
České vysoké učení technické v Praze

# Pojmenované argumenty (named/keyword arguments, *kwargs*)

keyword\_args.py – rozdíl obyčejných a pojmenovaných argumentů

```
1 def my_function(p1, p2, p3=10, p4=None): # p3, p4 are kwargs
2     result = p1 + p2
3     if p4 == 'expand':
4         result += p3
5     return result
6
7 if __name__ == '__main__':
8     r1 = my_function(3, 5) # p3, p4 -> default values
9     r2 = my_function(10, 20, p4='expand') # skipped p3 -> default value
10    r3 = my_function(10, 20, p4='expand', p3=30) # kwargs order not strict
```

# Pojmenované argumenty

keyword\_args\_2.py

```
1 def my_function(x=[0,0]):  
2     x[0] = x[0] + 1  
3     return x + [1]  
4  
5 a = my_function()  
6 b = my_function()  
7  
8 print(a, b)
```

# Pojmenované argumenty

keyword\_args\_2.py

```
1 def my_function(x=[0,0]):  
2     x[0] = x[0] + 1  
3     return x + [1]  
4  
5 a = my_function()  
6 b = my_function()  
7  
8 print(a, b)
```

Program vytiskne:

- (a) [1, 0, 1] [1, 0, 1]
- (b) [2, 0] [2, 0]
- (c) [1, 0, 1] [2, 0, 1]
- (d) dojde k chybě

# Potenciální nebezpečí

## Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

Python 3.6  
[\(known limitations\)](#)

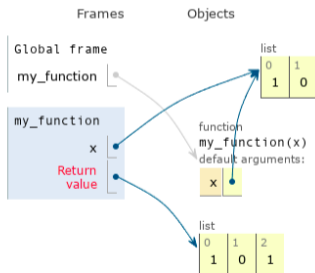
```
1 def my_function(x=[0,0]):  
2     x[0] = x[0] + 1  
3     return x + [1]  
4  
5 a = my_function()  
6 b = my_function()  
7  
8 print(a, b)
```

[Edit this code](#)

→ line that just executed  
→ next line to execute

Step 6 of 12

Print output (drag lower right corner to resize)



# Potenciální nebezpečí

## Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

Python 3.6  
[\(known limitations\)](#)

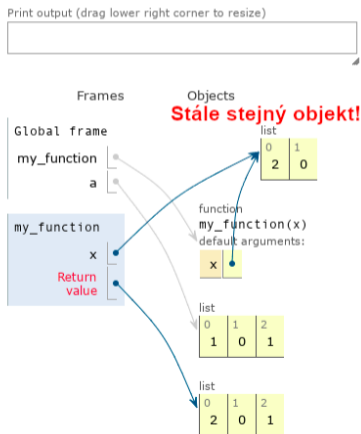
```
1 def my_function(x=[0,0]):  
2     x[0] = x[0] + 1  
3     return x + [1]  
4  
5 a = my_function()  
6 b = my_function()  
7  
8 print(a, b)
```

[Edit this code](#)

→ line that just executed  
→ next line to execute

Step 11 of 12

[Customize visualization](#)



## Možné řešení

```
1 def my_function(x=None):
2
3     if x is None:
4         x = [0, 0] # list created AFTER function call
5         # created in function scope, not global
6
7         x[0] = x[0] + 1
8         return x + [1]
9
10 a = my_function()
11 b = my_function()
12
13 print(a, b)
```

# Datový typ slovník (dictionary, `dict`)

## Slovník – `dict()`

- Datový kontejner pro párování
- Klíč (key) : hodnota (value)
- Iterable
- Indexování pomocí klíče
- Klíč – unikátní, *hashovatelný*
- [Online materiály ke slovníkům](#)



## Ukázka práce se slovníky

- Vytvoření nového slovníku
- Přidávání a odebírání dat
- For-cykly

# Ukázka práce se slovníky

## dictionary\_demo.py

```
1 d = {} # create empty dictionary
2
3 # adding new data:
4 d[0] = 'zero' # key is 0, value is 'zero'
5 d['a'] = 'hello' # key is 'a', value 'hello'
6 d['x'] = 'RPH'
7 d[('petr', 'stibinger')] = 'stibipet@fel.cvut.cz'
8
9 print(d) # will be ordered by insertion time
10 print(d['x'])
11 print(d.keys(), d.values()) # these methods return special objects
12
13 # iteration
14 for k in d: # only keys, have to use indexing to get values
15     print(f"KEY {k}, VALUE {d[k]}")
16
17 for k, val in d.items(): # dict.items generates tuples (key, value)
18     print(f"KEY {k}, VALUE {val}")
```

# Payoff matice jako slovník

## Python Tutor: Visualize code in [Python](#), [JavaScript](#), [C](#), [C++](#), and [Java](#)

```

Python 3.6
(known limitations)

1 pm = {}
2 pm['R', 'P'] = (0,1)
3 pm['P', 'S'] = (0,1)
4 pm['S', 'R'] = (0,1)
5 pm['P', 'R'] = (1,0)
6 pm['S', 'P'] = (1,0)
7 pm['R', 'S'] = (1,0)
8
9 → py_profit = pm['S', 'P'][0]
    
```

[Edit this code](#)

→ line that just executed  
 → next line to execute



<< First < Prev Next > Last >>

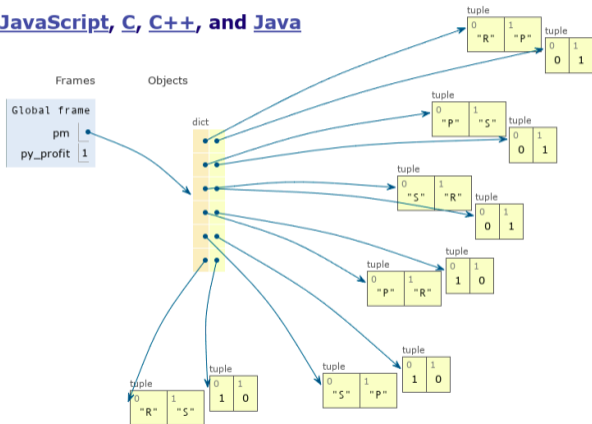
Done running (8 steps)

[Customize visualization](#)

**Warning:** Reloading this page loses all changes; customizations are NOT shared in URL

**Drag any heap object** to move it around the canvas.

Line style:



## Vlastnosti slovníku

- Pořadí podle času vložení (od Pythonu 3.7)
- Okamžitý přístup – na pořadí *obvykle* nezáleží
- V případě nutnosti **sorted()**

# Operátor `in`

for\_loop\_in.py - ukázka využití operátoru `in`

```
1 my_list = ['dog', 'cat', 'bird']
2 my_dictionary = {'dog' : 'bark', 'cat' : 'meow'}
3
4 # syntax we already know and love
5 for animal in my_list:
6     print(animal)
7
8 # searching in containers
9 if 'frog' in my_list:
10     # použije se list.__contains__(obj):
11     print('found a frog')
12
13 if 'dog' in my_dictionary: # prohledava klice
14     print('found a dog')
```

# Skládání objektů (*kompozice*) – rps\_player\_memory.py

```
1 class Memory:
2
3     def __init__(self, size):
4         self.size = size
5         self.data = []
6
7     def add(self, value):
8         self.data.append(value)
9         if len(self.data) > self.size:
10            del self.data[0]
11
12     def get_most_frequent(self):
13         return max(self.data, key=self.data.count)
14
15 class MyPlayer:
16
17     def __init__(self):
18         self.my_moves = Memory(100)
19         self.opp_moves = Memory(100)
20
21     def play(self):
22         return 'R'
23
24     def record_last_moves(self, my_m, opp_m):
25         self.my_moves.add(my_m)
26         self.opp_moves.add(opp_m)
27         print(f"My opponent plays mostly {self.opp_moves.get_most_frequent()}!")
```

# Skládání objektů (*kompozice*) – rps\_player\_memory.py

```
1  class Memory:
2
3      def __init__(self, size):
4          self.size = size
5          self.data = []
6
7      def add(self, value):
8          self.data.append(value)
9          if len(self.data) > self.size:
10             del self.data[0]
11
12      def get_most_frequent(self):
13          return max(self.data, key=self.data.count)
14
15  class MyPlayer:
16
17      def __init__(self):
18          self.my_moves = Memory(100)
19          self.opp_moves = Memory(100)
20
21      def play(self):
22          return 'R'
23
24      def record_last_moves(self, my_m, opp_m):
25          self.my_moves.add(my_m)
26          self.opp_moves.add(opp_m)
27          print(f"My opponent plays mostly {self.opp_moves.get_most_frequent()}!")
```

Operace, které se přímo netýkají hry

# Skládání objektů (*kompozice*) – rps\_player\_memory.py

```
1 class Memory:
2
3     def __init__(self, size):
4         self.size = size
5         self.data = []
6
7     def add(self, value):
8         self.data.append(value)
9         if len(self.data) > self.size:
10            del self.data[0]
11
12     def get_most_frequent(self):
13         return max(self.data, key=self.data.count)
```

Operace, které se přímo netýkají hry

```
15 class MyPlayer:
16
17     def __init__(self):
18         self.my_moves = Memory(100)
19         self.opp_moves = Memory(100)
20
21     def play(self):
22         return 'R'
23
24     def record_last_moves(self, my_m, opp_m):
25         self.my_moves.add(my_m)
26         self.opp_moves.add(opp_m)
27         print(f"My opponent plays mostly {self.opp_moves.get_most_frequent()}!")
```

Interakce s hrou dle specifikace



# Skládání objektů (*kompozice*) – rps\_player\_memory.py

```
1 class Memory:
2
3     def __init__(self, size):
4         self.size = size
5         self.data = []
6
7     def add(self, value):
8         self.data.append(value)
9         if len(self.data) > self.size:
10            del self.data[0]
11
12     def get_most_frequent(self):
13         return max(self.data, key=self.data.count)
```

Operace, které se přímo netýkají hry

```
15 class MyPlayer:
16
17     def __init__(self):
18         self.my_moves = Memory(100)
19         self.opp_moves = Memory(100)
20
21     def play(self):
22         return 'R'
23
24     def record_last_moves(self, my_m, opp_m):
25         self.my_moves.add(my_m)
26         self.opp_moves.add(opp_m)
27         print(f"My opponent plays mostly {self.opp_moves.get_most_frequent()}!")
```

Interakce s hrou dle specifikace

Ve vlastní třídě používáme jinou vlastní třídu

# Dědičnost

ConstantPlayer, DummyPlayer, RandomPlayer...

- Stejné rozhraní
  - `play()`
  - `record_last_moves()`
- Rozdílná vnitřní logika
  - Využití paměti
  - Způsob výběru tahu
  - “Umělá inteligence”

# Dědičnost – abstrakce a odvození

```
1 class BasePlayer:
2
3     OPTIONS=('R', 'P', 'S')
4
5     def __init__(self):
6         self.my_moves = Memory(100)
7         self.opp_moves = Memory(100)
8
9     def play(self):
10        my_move = self.select_move()
11        return my_move
12
13    def record_last_moves(self, my_m, opp_m):
14        self.my_moves.add(my_m)
15        self.opp_moves.add(opp_m)
16
17    def select_move(self):
18        raise NotImplementedError
19
20    def __str__(self):
21        return self.__class__.__name__
```

# Dědičnost – abstrakce a odvození

```
1 class BasePlayer:
2
3     OPTIONS=('R', 'P', 'S')
4
5     def __init__(self):
6         self.my_moves = Memory(100)
7         self.opp_moves = Memory(100)
8
9     def play(self):
10        my_move = self.select_move()
11        return my_move
12
13    def record_last_moves(self, my_m, opp_m):
14        self.my_moves.add(my_m)
15        self.opp_moves.add(opp_m)
16
17    def select_move(self):
18        raise NotImplementedError
19
20    def __str__(self):
21        return self.__class__.__name__
```

Základní třída, společné vlastnosti

# Dědičnost – abstrakce a odvození

```
1 class BasePlayer:
2
3     OPTIONS=('R', 'P', 'S')
4
5     def __init__(self):
6         self.my_moves = Memory(100)
7         self.opp_moves = Memory(100)
8
9     def play(self):
10        my_move = self.select_move()
11        return my_move
12
13    def record_last_moves(self, my_m, opp_m):
14        self.my_moves.add(my_m)
15        self.opp_moves.add(opp_m)
16
17    def select_move(self):
18        raise NotImplementedError
19
20    def __str__(self):
21        return self.__class__.__name__
```

Základní třída, společné vlastnosti

Každý hráč umí play()...

# Dědičnost – abstrakce a odvození

```
1 class BasePlayer:
2
3     OPTIONS=('R', 'P', 'S')
4
5     def __init__(self):
6         self.my_moves = Memory(100)
7         self.opp_moves = Memory(100)
8
9     def play(self):
10        my_move = self.select_move()
11        return my_move
12
13    def record_last_moves(self, my_m, opp_m):
14        self.my_moves.add(my_m)
15        self.opp_moves.add(opp_m)
16
17    def select_move(self):
18        raise NotImplementedError
19
20    def __str__(self):
21        return self.__class__.__name__
```

Základní třída, společné vlastnosti

Každý hráč umí play()...

Ale liší se ve způsobu výběru!

# Dědičnost – abstrakce a odvození

```
1 class BasePlayer:
2
3     OPTIONS=('R', 'P', 'S')
4
5     def __init__(self):
6         self.my_moves = Memory(100)
7         self.opp_moves = Memory(100)
8
9     def play(self):
10        my_move = self.select_move()
11        return my_move
12
13    def record_last_moves(self, my_m, opp_m):
14        self.my_moves.add(my_m)
15        self.opp_moves.add(opp_m)
16
17    def select_move(self):
18        raise NotImplementedError
19
20    def __str__(self):
21        return self.__class__.__name__
```

Základní třída, společné vlastnosti

Každý hráč umí play()...

Ale liší se ve způsobu výběru!

(Úmyslně) skončí s chybou

# Dědičnost – abstrakce a odvození

```
1 class BasePlayer:
2
3     OPTIONS=('R', 'P', 'S')
4
5     def __init__(self):
6         self.my_moves = Memory(100)
7         self.opp_moves = Memory(100)
8
9     def play(self):
10        my_move = self.select_move()
11        return my_move
12
13    def record_last_moves(self, my_m, opp_m):
14        self.my_moves.add(my_m)
15        self.opp_moves.add(opp_m)
16
17    def select_move(self):
18        raise NotImplementedError
19
20    def __str__(self):
21        return self.__class__.__name__
```

Základní třída, společné vlastnosti

Každý hráč umí play()...

Ale liší se ve způsobu výběru!

(Úmyslně) skončí s chybou

Implementaci poskytnete odvozená třída



# Dědičnost – abstrakce a odvození

```
1 class BasePlayer:
2
3     OPTIONS=('R', 'P', 'S')
4
5     def __init__(self):
6         self.my_moves = Memory(100)
7         self.opp_moves = Memory(100)
8
9     def play(self):
10        my_move = self.select_move()
11        return my_move
12
13    def record_last_moves(self, my_m, opp_m):
14        self.my_moves.add(my_m)
15        self.opp_moves.add(opp_m)
16
17    def select_move(self):
18        raise NotImplementedError
19
20    def __str__(self):
21        return self.__class__.__name__
```

```
1 class RandomPlayer(BasePlayer):
2
3     def select_move(self):
4         return random.choice(self.OPTIONS)
```

# Dědičnost – abstrakce a odvození

```
1 class BasePlayer:
2
3     OPTIONS=('R', 'P', 'S')
4
5     def __init__(self):
6         self.my_moves = Memory(100)
7         self.opp_moves = Memory(100)
8
9     def play(self):
10        my_move = self.select_move()
11        return my_move
12
13    def record_last_moves(self, my_m, opp_m):
14        self.my_moves.add(my_m)
15        self.opp_moves.add(opp_m)
16
17    def select_move(self):
18        raise NotImplementedError
19
20    def __str__(self):
21        return self.__class__.__name__
```

```
1 class RandomPlayer(BasePlayer):
2
3     def select_move(self):
4         return random.choice(self.OPTIONS)
```

Třída RandomPlayer dědí od BasePlayer

# Dědičnost – abstrakce a odvození

```
1 class BasePlayer:
2
3     OPTIONS=('R', 'P', 'S')
4
5     def __init__(self):
6         self.my_moves = Memory(100)
7         self.opp_moves = Memory(100)
8
9     def play(self):
10        my_move = self.select_move()
11        return my_move
12
13    def record_last_moves(self, my_m, opp_m):
14        self.my_moves.add(my_m)
15        self.opp_moves.add(opp_m)
16
17    def select_move(self):
18        raise NotImplementedError
19
20    def __str__(self):
21        return self.__class__.__name__
```

```
1 class RandomPlayer(BasePlayer):
2
3     def select_move(self):
4         return random.choice(self.OPTIONS)
```

Třída RandomPlayer dědí od BasePlayer

Python hledá implementaci podle MRO

# Díky za pozornost

## Prostor pro dotazy

Příště – množiny, list comprehension, generátory, profilování

Na cvičení – třetí testík, začátek řešení úlohy reversi