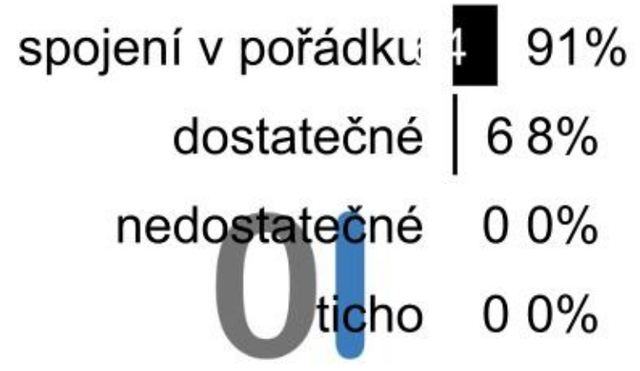
Python, základní kameny až skály III

Tomáš Svoboda

B4B33RPH, 2020-10-27

live coding sessions

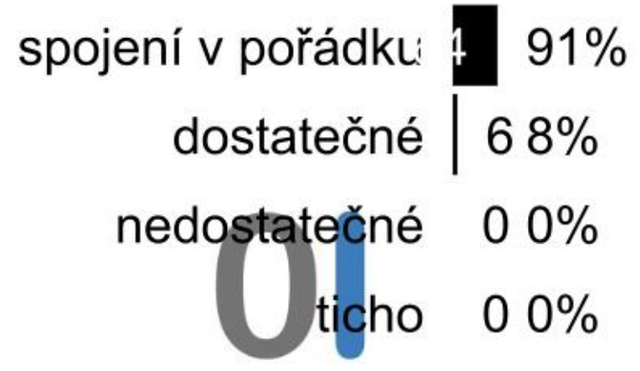


Python, základní kameny až skály III

Tomáš Svoboda

B4B33RPH, 2020-10-27

live coding sessions



Dnes …

Dnes …

- set, frozen set

Dnes …

- set, frozen set
- list comprehensions (generátorová notace)

Dnes …

- set, frozen set
- list comprehensions (generátorová notace)
- malá ukázka grafického výstupu a měření efektivity

Dnes …

- set, frozen set
- list comprehensions (generátorová notace)
- malá ukázka grafického výstupu a měření efektivity
- logické funkce

Dnes …

- set, frozen set
- list comprehensions (generátorová notace)
- malá ukázka grafického výstupu a měření efektivity
- logické funkce
- generátory

slajdy nejsou vše

- klíčové podněty
- kódy z přednášky na hraní - refaktorujte, zlepšujte, přidávejte funkcionalitu
- http://cw.fel.cvut.cz/wiki/courses/b4b33rph/literatura
- https://stackoverflow.com,
- páteční odpolední cvičení
- Programujte!

množina - set, frozenset

```
1 >>> a = set([1,2,3,3])
2 >>> print(a)
```

A: {1,2,3,3}

B: {1,2,3}

C: {3,3}

množina - set, frozenset

```
1 >>> a = set([1,2,3,3])
2 >>> print(a)
```

A: {1,2,3,3}

B: {1,2,3}

C: {3,3}

množina - set, frozenset

```
1 >>> a = set([1,2,3,3])
2 >>> print(a)
```

A: {1,2,3,3}
B: {1,2,3}
C: {3,3}

```
3 >>> b = set([2,3,4])
4 >>> a | b
```

A: {1,2,3,4}
B: {1,2,2,3,3,3,4}
C: {3,3,3}

množina - set, frozenset

```
1 >>> a = set([1,2,3,3])
2 >>> print(a)
```

A: {1,2,3,3}
B: {1,2,3}
C: {3,3}

```
3 >>> b = set([2,3,4])
4 >>> a | b
```

A: {1,2,3,4}
B: {1,2,2,3,3,3,4}
C: {3,3,3}

množina - set, frozenset

```
1 >>> a = set([1,2,3,3])
2 >>> print(a)
```

A: {1,2,3,3}
B: {1,2,3}
C: {3,3}

```
3 >>> b = set([2,3,4])
4 >>> a | b
```

A: {1,2,3,4}
B: {1,2,2,3,3,3,4}
C: {3,3,3}

```
5 >>> a & b
```

A: {2,3,4}
B: {2,3}

množina - set, frozenset

```
1 >>> a = set([1,2,3,3])
2 >>> print(a)
```

A: {1,2,3,3}
B: {1,2,3}
C: {3,3}

```
3 >>> b = set([2,3,4])
4 >>> a | b
```

A: {1,2,3,4}
B: {1,2,2,3,3,3,4}
C: {3,3,3}

```
5 >>> a & b
```

A: {2,3,4}
B: {2,3}

a = {1,2,3} [ ] ( )

množina - set, frozenset

```
1 >>> a = set([1,2,3,3])
2 >>> print(a)
```

A: {1,2,3,3}
B: {1,2,3}
C: {3,3}

```
3 >>> b = set([2,3,4])
4 >>> a | b
```

A: {1,2,3,4}
B: {1,2,2,3,3,3,4}
C: {3,3,3}

```
5 >>> a & b
```

A: {2,3,4}
B: {2,3}

set - mutable

frozenset - immutable

https://docs.python.org/3.8/library/stdtypes.html#set-types-set-frozenset

List comprehensions

- list comprehensions - kompaktní vytvoření seznamu bez explicitní for smyčky
- někdy se říka generátorová notace, ale *generátor* má specifický význam
- `[x**2 for x in range(-10,10)]`
- `[x**(0.5) for x in range(-10, 10) if x>0]`

množina - set, frozenset

```
1 >>> a = set([1,2,3,3])
2 >>> print(a)
```

A: {1,2,3,3}
B: {1,2,3}
C: {3,3}

```
3 >>> b = set([2,3,4])
4 >>> a | b
```

A: {1,2,3,4}
B: {1,2,2,3,3,3,4}
C: {3,3,3}

```
5 >>> a & b
```

A: {2,3,4}
B: {2,3}

set - mutable

frozenset - immutable

https://docs.python.org/3.8/library/stdtypes.html#set-types-set-frozenset

množina - set, frozenset

```
1 >>> a = set([1,2,3,3])
2 >>> print(a)
```

A: {1,2,3,3}
B: {1,2,3}
C: {3,3}

```
3 >>> b = set([2,3,4])
4 >>> a | b
```

A: {1,2,3,4}
B: {1,2,2,3,3,3,4}
C: {3,3,3}

```
5 >>> a & b
```

A: {2,3,4}
B: {2,3}

a = {1,2,3} [ ] ( )

množina - set, frozenset

```
1 >>> a = set([1,2,3,3])
2 >>> print(a)
```

A: {1,2,3,3}
B: {1,2,3}
C: {3,3}

4, 3, 2

```
3 >>> b = set([2,3,4])
4 >>> a | b
```

A: {1,2,3,4}
B: {1,2,2,3,3,3,4}
C: {3,3,3}

```
5 >>> a & b
```

A: {2,3,4}
B: {2,3}

a = {1,2,3} [ ] ( )

~~a[0]~~

množina - set, frozenset

```
1 >>> a = set([1,2,3,3])
2 >>> print(a)
```

A: {1,2,3,3}
B: {1,2,3}
C: {3,3}

4, 3, 2

```
3 >>> b = set([2,3,4])
4 >>> a | b
```

A: {1,2,3,4}
B: {1,2,2,3,3,3,4}
C: {3,3,3}

```
5 >>> a & b
```

A: {2,3,4}
B: {2,3}

a = {1,2,3} [ ] ( )

~~a[0]~~

množina - set, frozenset

```
1 >>> a = set([1,2,3,3])
2 >>> print(a)
```

A: {1,2,3,3}
B: {1,2,3}
C: {3,3}

4, 3, 2

```
3 >>> b = set([2,3,4])
4 >>> a | b
```

A: {1,2,3,4}
B: {1,2,2,3,3,3,4}
C: {3,3,3}

```
5 >>> a & b
```

A: {2,3,4}
B: {2,3}

a = {1,2,3} [ ] ( )

~~a[0]~~

množina - set, frozenset

```
1 >>> a = set([1,2,3,3])
2 >>> print(a)
```

A: {1,2,3,3}
B: {1,2,3}
C: {3,3}

```
3 >>> b = set([2,3,4])
4 >>> a | b
```

A: {1,2,3,4}
B: {1,2,2,3,3,3,4}
C: {3,3,3}

```
5 >>> a & b
```

A: {2,3,4}
B: {2,3}

set - mutable

frozenset - immutable

https://docs.python.org/3.8/library/stdtypes.html#set-types-set-frozenset

List comprehensions

- list comprehensions - kompaktní vytvoření seznamu bez explicitní for smyčky
- někdy se říka generátorová notace, ale *generátor* má specifický význam
- `[x**2 for x in range(-10,10)]`
- `[x**(0.5) for x in range(-10, 10) if x>0]`

List comprehensions

- list comprehensions - kompaktní vytvoření seznamu bez explicitní for smyčky
- někdy se říka generátorová notace, ale *generátor* má specifický význam
- `[x**2 for x in range(-10,10)]`
- `[x**(0.5) for x in range(-10, 10) if x>0]`

birthday problem

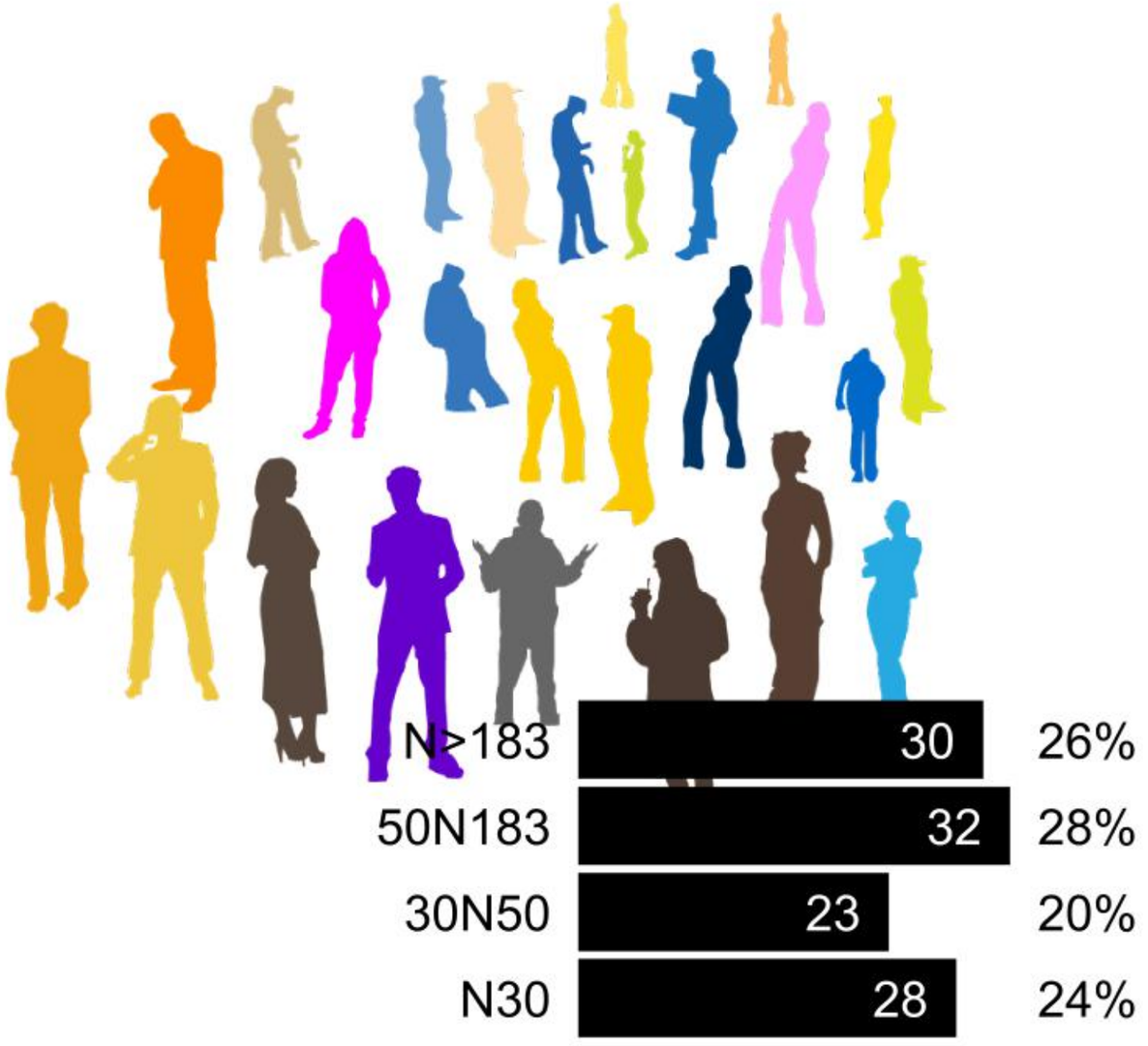
- Skupina N osob
- Jak velká je pravděpodobnost že alespoň jedno datum narození není unikátní?
- Pro jak velké N začne být pravděpodobnější, že alespoň jedny narozeniny jsou společné?

birthday problem

$N > 1$, 365 dní v roce

- Skupina N osob
- Jak velká je pravděpodobnost že alespoň jedno datum narození není unikátní?
- Pro jak velké N začne být pravděpodobnější, že alespoň jedny narozeniny jsou společné?

$P(N) > 0.5$



Hlavní zkoušecí smyčka

```
1 def compute_stats(total_trials, matching_fcn):
2     prob_of_matching_dates = {}
3     for group_size in range(2,183):
4         prob_of_matching_dates[group_size] = 0.0
5         for trial in range(total_trials):
6             if matching_fcn(group_size):
7                 prob_of_matching_dates[group_size] += 1/
total_trials
8     return prob_of_matching_dates
```

`matching_fcn` je odkaz na funkci, která vrací True, pokud v náhodně vygenerovaném seznamu narozenin je shoda.

[N]: P(N)

Hlavní zkoušecí smyčka

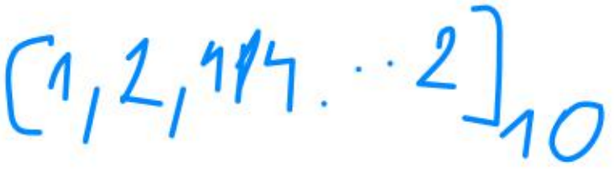
```
1 def compute_stats(total_trials, matching_fcn):
2     prob_of_matching_dates = {}
3     for group_size in range(2,183):
4         prob_of_matching_dates[group_size] = 0.0
5         for trial in range(total_trials):
6             if matching_fcn(group_size):
7                 prob_of_matching_dates[group_size] += 1/
total_trials
8     return prob_of_matching_dates
```

`matching_fcn` je odkaz na funkci, která vrací True, pokud v náhodně vygenerovaném seznamu narozenin je shoda.

Líné řešení

```
1 def is_matching_date(group_size):
2     days = [random.randint(1,365) for i in range(group_size)]
3     return len(days) != len(set(days))
```

Líné řešení



```
1 def is_matching_date(group_size):
2     days = [random.randint(1,365) for i in range(group_size)]
3     return len(days) != len(set(days))
```

Líné řešení

```
1 def is_matching_date(group_size):
2     days = [random.randint(1,365) for i in range(group_size)]
3     return len(days) != len(set(days))
```

Jak měřit rychlost

```
1 start_time = time.time()
2 probs_of_matching = compute_stats(100, is_matching_date)
3 elapsed_time = time.time() - start_time
4 print('Elapsed time:', elapsed_time)
```

Líné řešení

```
1 def is_matching_date(group_size):
2     days = [random.randint(1,365) for i in range(group_size)]
3     return len(days) != len(set(days))
```

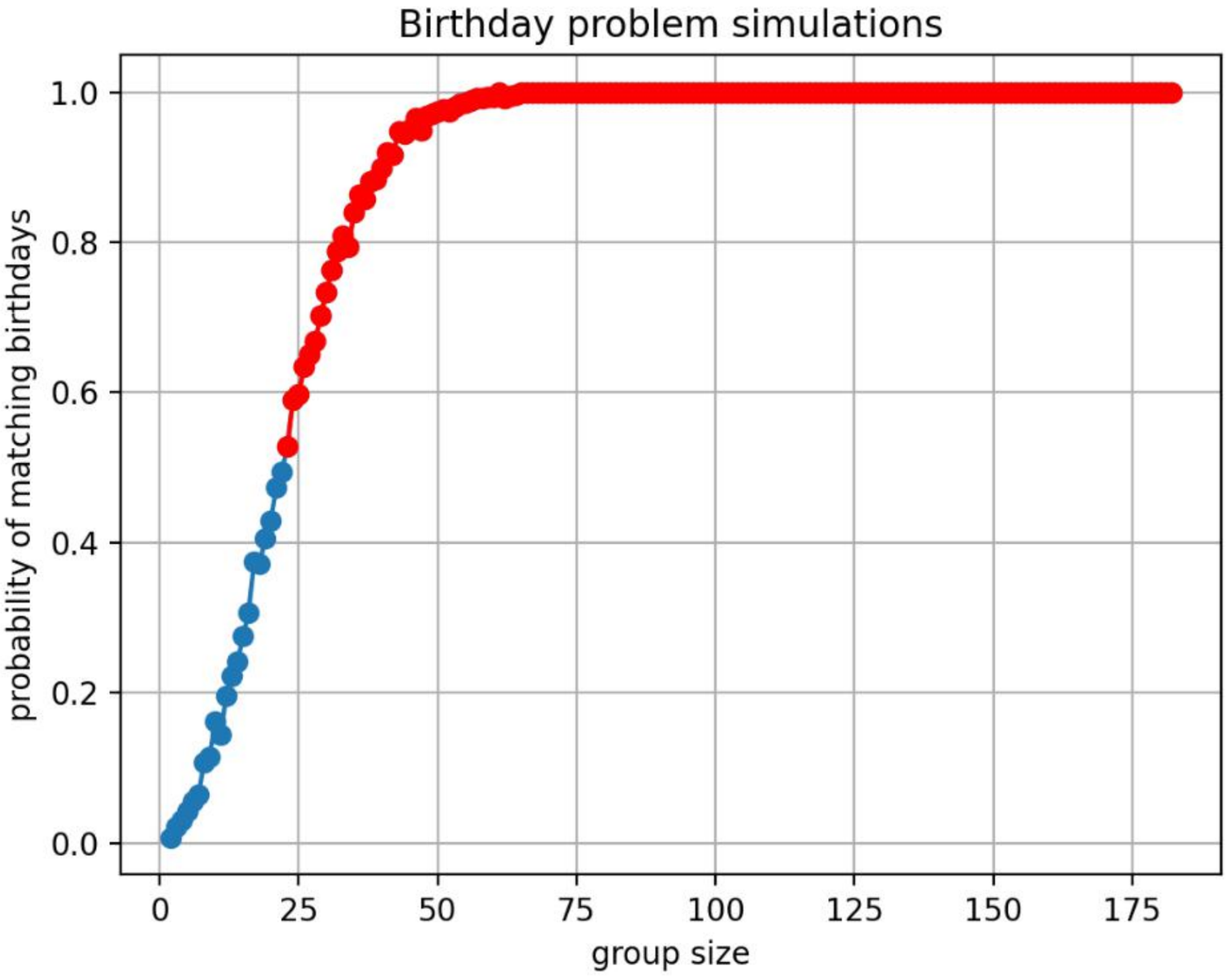
Jak měřit rychlost

import time

```
1 start_time = time.time()
2 probs_of_matching = compute_stats(100, is_matching_date)
3 elapsed_time = time.time() - start_time
4 print('Elapsed time:', elapsed_time)
```

matplotlib.pyplot

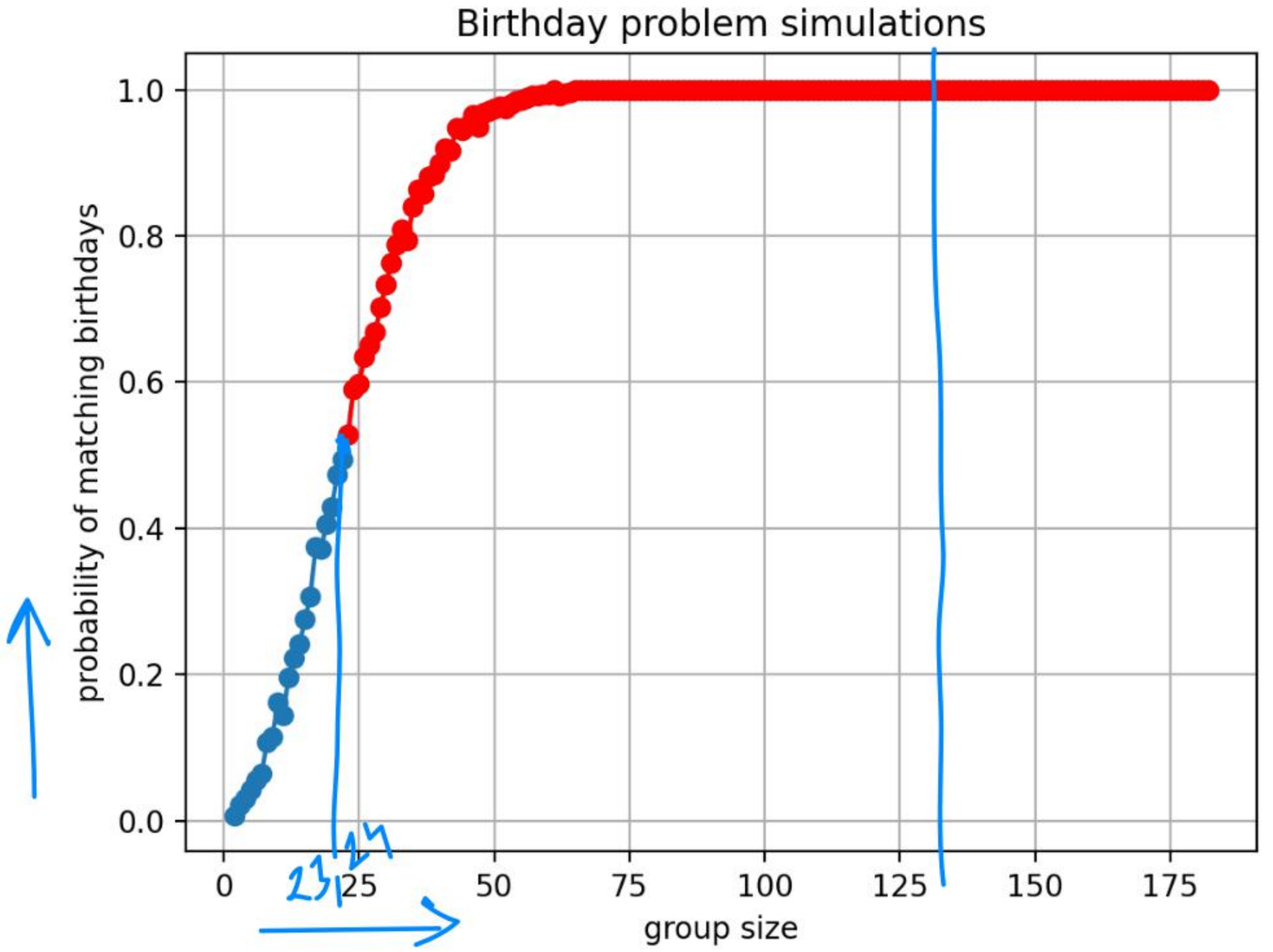
`birthday_problem.py`



matplotlib.pyplot

`birthday_problem.py`

N<30



Líné řešení

```
1 def is_matching_date(group_size):
2     days = [random.randint(1,365) for i in range(group_size)]
3     return len(days) != len(set(days))
```

Jak měřit rychlost

import time

```
1 start_time = time.time()
2 probs_of_matching = compute_stats(100, is_matching_date)
3 elapsed_time = time.time() - start_time
4 print('Elapsed time:', elapsed_time)
```

Líné řešení

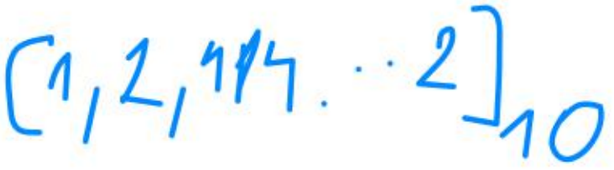
```
1 def is_matching_date(group_size):
2     days = [random.randint(1,365) for i in range(group_size)]
3     return len(days) != len(set(days))
```

Jak měřit rychlost

import time

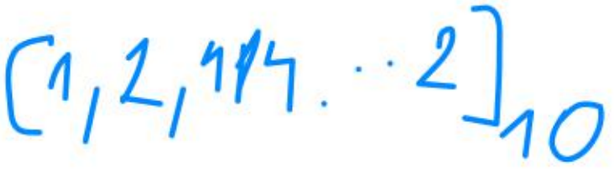
```
1 start_time = time.time()
2 probs_of_matching = compute_stats(100, is_matching_date)
3 elapsed_time = time.time() - start_time
4 print('Elapsed time:', elapsed_time)
```

Líné řešení



```
1 def is_matching_date(group_size):
2     days = [random.randint(1,365) for i in range(group_size)]
3     return len(days) != len(set(days))
```

Líné řešení



```
1 def is_matching_date(group_size):
2     days = [random.randint(1,365) for i in range(group_size)]
3     return len(days) != len(set(days))
```

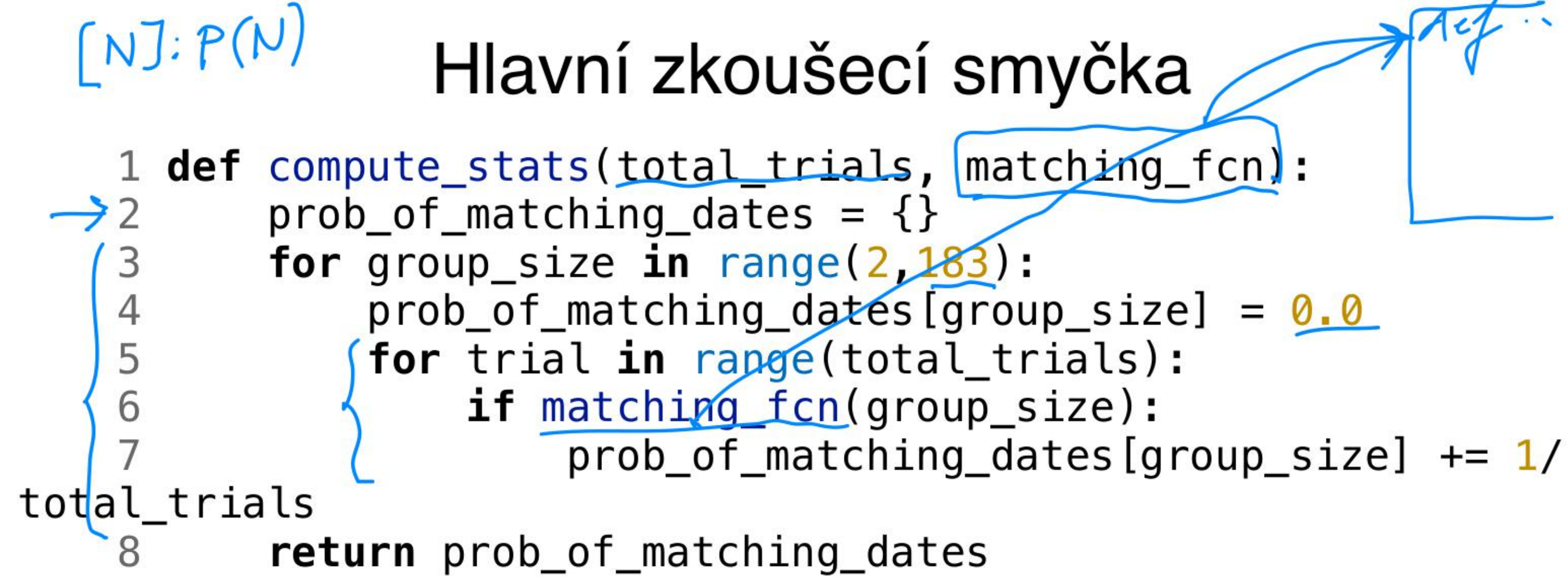
[N]: P(N)

Hlavní zkoušecí smyčka

```
1 def compute_stats(total_trials, matching_fcn):
2     prob_of_matching_dates = {}
3     for group_size in range(2,183):
4         prob_of_matching_dates[group_size] = 0.0
5         for trial in range(total_trials):
6             if matching_fcn(group_size):
7                 prob_of_matching_dates[group_size] += 1/
total_trials
8     return prob_of_matching_dates
```

`matching_fcn` je odkaz na funkci, která vrací True, pokud v náhodně vygenerovaném seznamu narozenin je shoda.

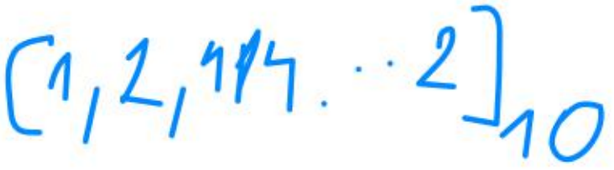
Hlavní zkoušecí smyčka



```
1 def compute_stats(total_trials, matching_fcn):
2     prob_of_matching_dates = {}
3     for group_size in range(2,183):
4         prob_of_matching_dates[group_size] = 0.0
5         for trial in range(total_trials):
6             if matching_fcn(group_size):
7                 prob_of_matching_dates[group_size] += 1/
total_trials
8     return prob_of_matching_dates
```

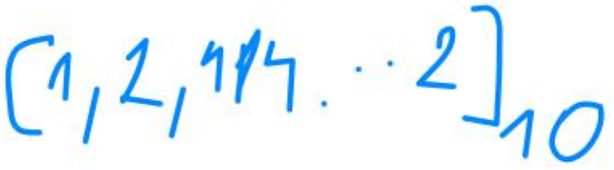
`matching_fcn` je odkaz na funkci, která vrací True, pokud v náhodně vygenerovaném seznamu narozenin je shoda.

Líné řešení



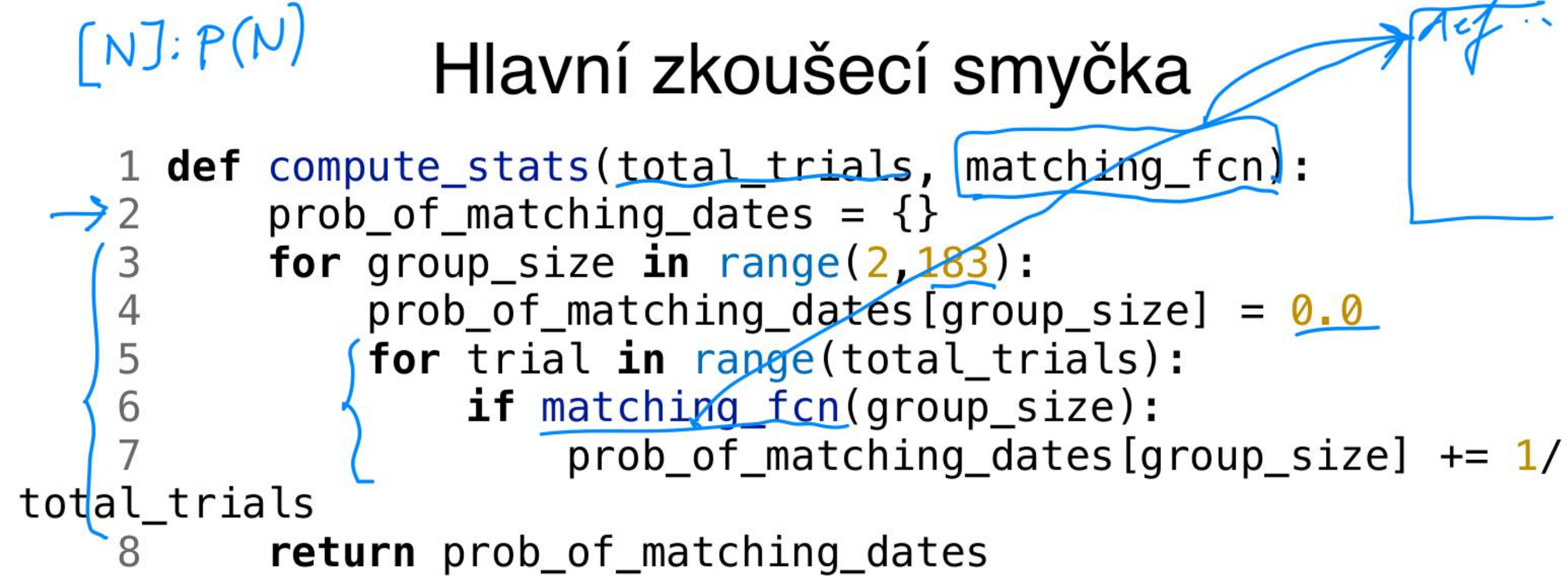
```
1 def is_matching_date(group_size):
2     days = [random.randint(1,365) for i in range(group_size)]
3     return len(days) != len(set(days))
```

Líné řešení



```
1 def is_matching_date(group_size):
2     days = [random.randint(1,365) for i in range(group_size)]
3     return len(days) != len(set(days))
```

Hlavní zkoušecí smyčka



```
1 def compute_stats(total_trials, matching_fcn):
2     prob_of_matching_dates = {}
3     for group_size in range(2,183):
4         prob_of_matching_dates[group_size] = 0.0
5         for trial in range(total_trials):
6             if matching_fcn(group_size):
7                 prob_of_matching_dates[group_size] += 1/
total_trials
8     return prob_of_matching_dates
```

`matching_fcn` je odkaz na funkci, která vrací True, pokud v náhodně vygenerovaném seznamu narozenin je shoda.

Hlavní zkoušecí smyčka

```
1 def compute_stats(total_trials, matching_fcn):
2     prob_of_matching_dates = {}
3     for group_size in range(2,183):
4         prob_of_matching_dates[group_size] = 0.0
5         for trial in range(total_trials):
6             if matching_fcn(group_size):
7                 prob_of_matching_dates[group_size] += 1/
total_trials
8     return prob_of_matching_dates
```

`matching_fcn` je odkaz na funkci, která vrací True, pokud v náhodně vygenerovaném seznamu narozenin je shoda.

Hlavní zkoušecí smyčka

```
1 def compute_stats(total_trials, matching_fcn):
2     prob_of_matching_dates = {}
3     for group_size in range(2,183):
4         prob_of_matching_dates[group_size] = 0.0
5         for trial in range(total_trials):
6             if matching_fcn(group_size):
7                 prob_of_matching_dates[group_size] += 1/
total_trials
8     return prob_of_matching_dates
```

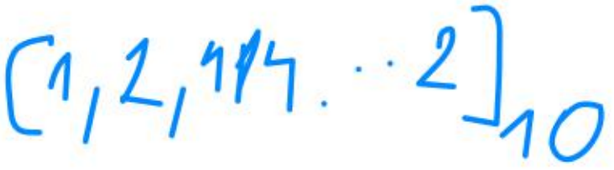
`matching_fcn` je odkaz na funkci, která vrací True, pokud v náhodně vygenerovaném seznamu narozenin je shoda.

Hlavní zkoušecí smyčka

```
1 def compute_stats(total_trials, matching_fcn):
2     prob_of_matching_dates = {}
3     for group_size in range(2,183):
4         prob_of_matching_dates[group_size] = 0.0
5         for trial in range(total_trials):
6             if matching_fcn(group_size):
7                 prob_of_matching_dates[group_size] += 1/
total_trials
8     return prob_of_matching_dates
```

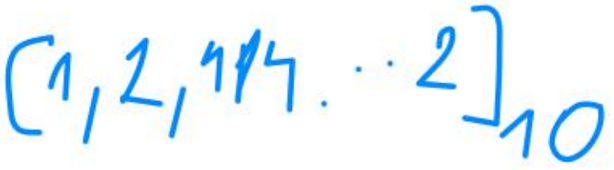
`matching_fcn` je odkaz na funkci, která vrací True, pokud v náhodně vygenerovaném seznamu narozenin je shoda.

Líné řešení



```
1 def is_matching_date(group_size):
2     days = [random.randint(1,365) for i in range(group_size)]
3     return len(days) != len(set(days))
```

Líné řešení



```
1 def is_matching_date(group_size):
2     days = [random.randint(1,365) for i in range(group_size)]
3     return len(days) != len(set(days))
```

Líné řešení

```
1 def is_matching_date(group_size):
2     days = [random.randint(1,365) for i in range(group_size)]
3     return len(days) != len(set(days))
```

Jak měřit rychlost

import time

```
1 start_time = time.time()
2 probs_of_matching = compute_stats(100, is_matching_date)
3 elapsed_time = time.time() - start_time
4 print('Elapsed time:', elapsed_time)
```

Líné řešení

```
1 def is_matching_date(group_size):
2     days = [random.randint(1,365) for i in range(group_size)]
3     return len(days) != len(set(days))
```

Jak měřit rychlost

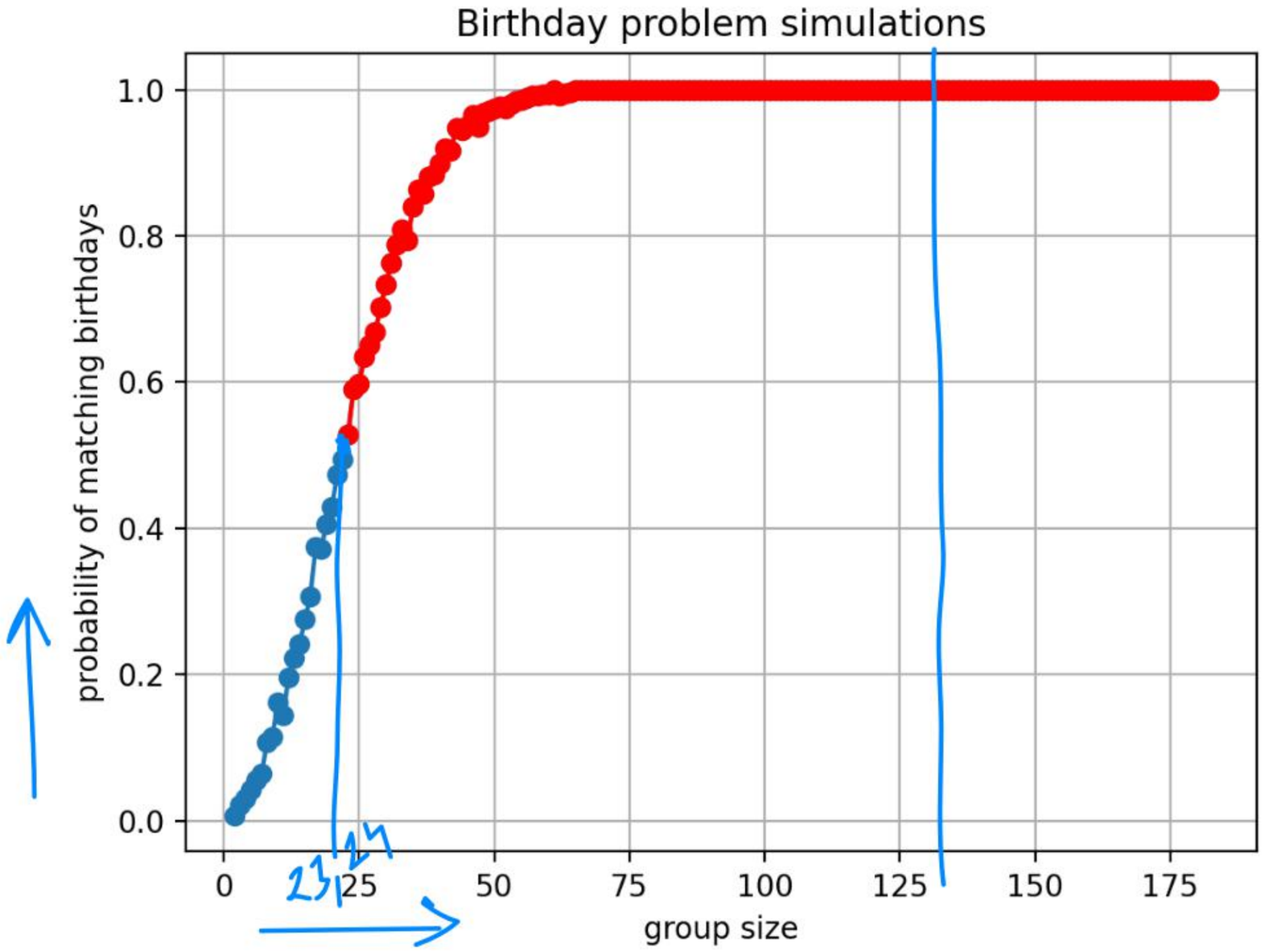
import time

```
1 start_time = time.time()
2 probs_of_matching = compute_stats(100, is_matching_date)
3 elapsed_time = time.time() - start_time
4 print('Elapsed time:', elapsed_time)
```

matplotlib.pyplot

`birthday_problem.py`

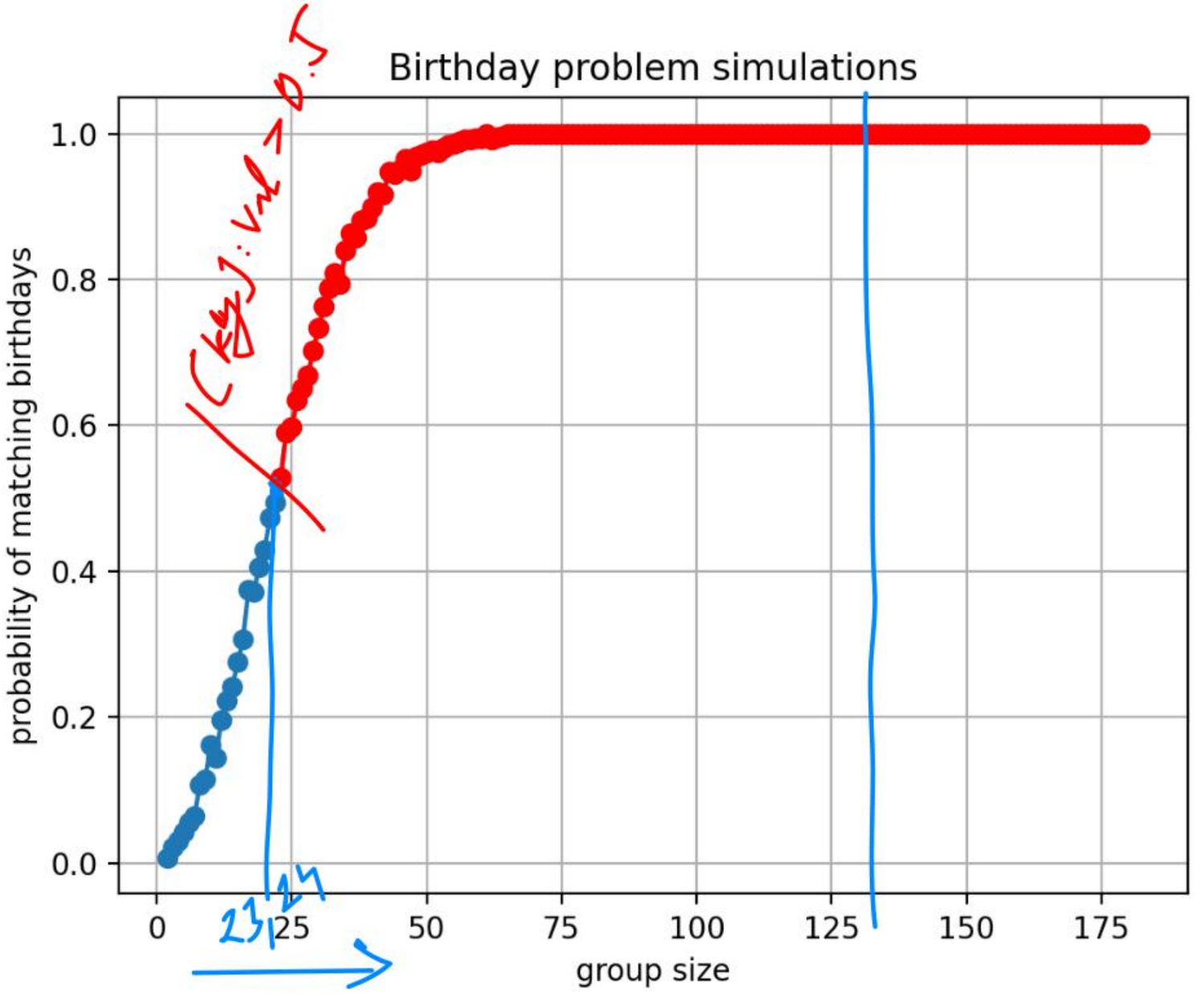
N<30



matplotlib.pyplot

birthday_problem.py

N<30



Líné řešení je líné

Líné řešení je líné

- potřebujeme počítat vše?

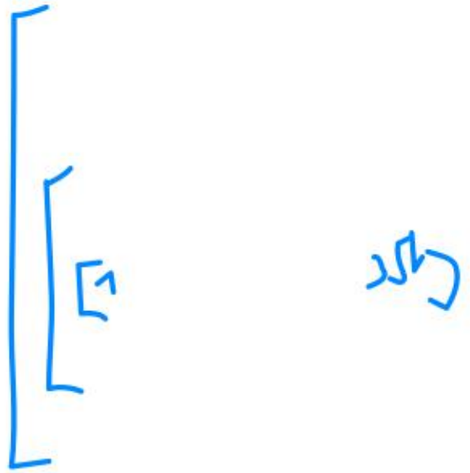
Líné řešení je líné

- potřebujeme počítat vše?
- stačí první případ společených narozenin

Líné řešení je líné

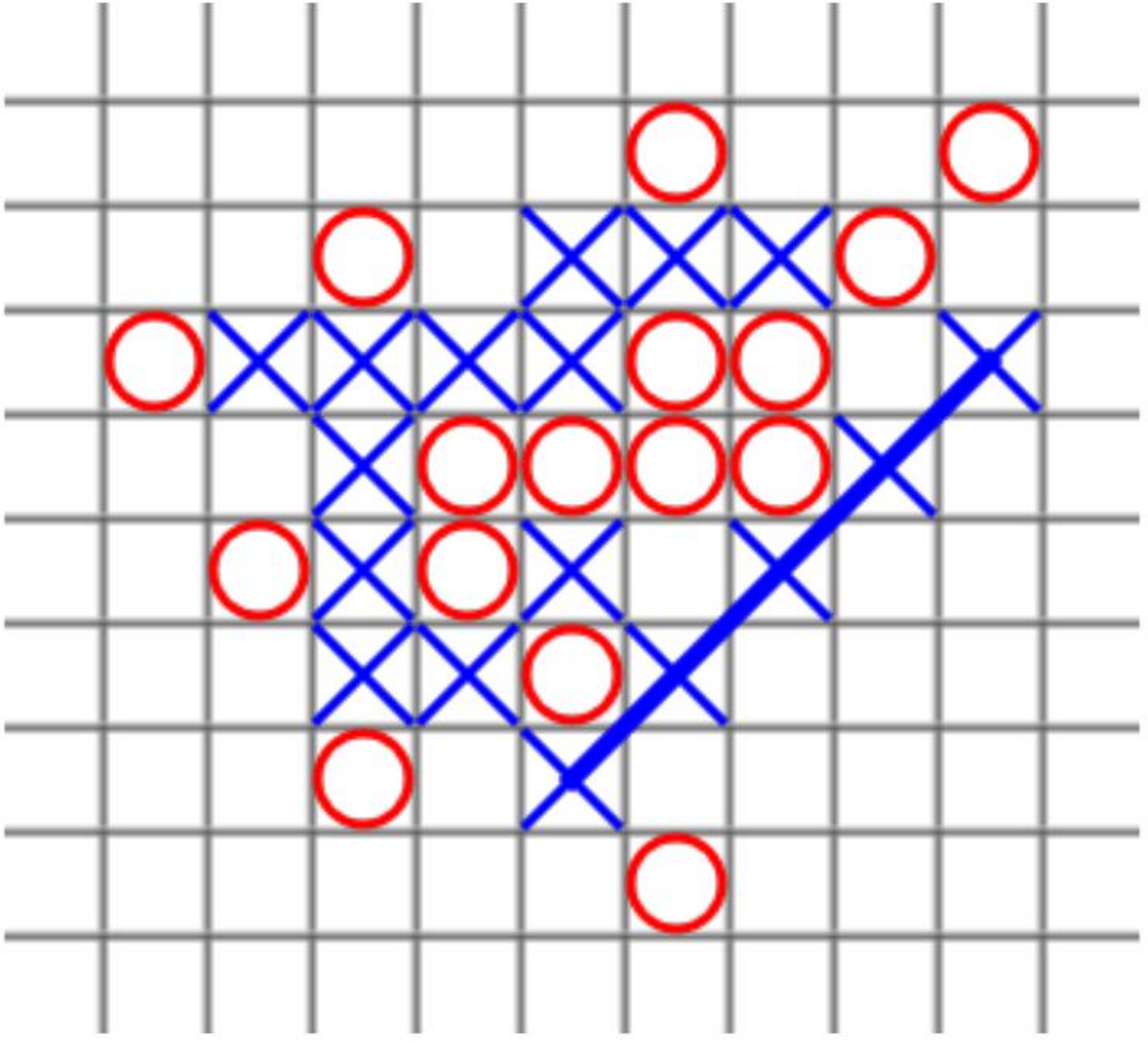
- potřebujeme počítat vše?
- stačí první případ společených narozenin
- algoritmus!

Líné řešení je líné



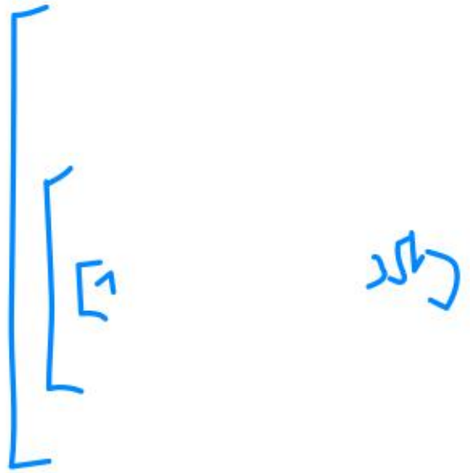
- potřebujeme počítat vše?
- stačí první případ společených narozenin
- algoritmus!

Piškvorky



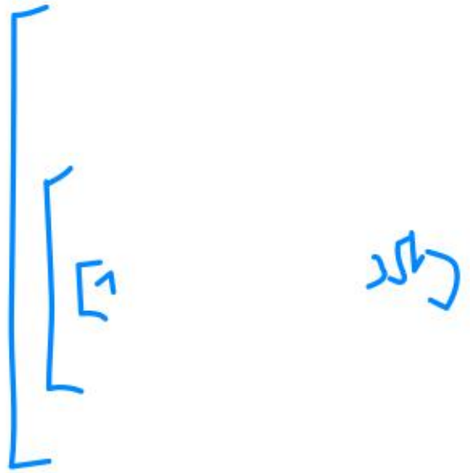
https://cs.wikipedia.org/wiki/Piškvorky

Líné řešení je líné



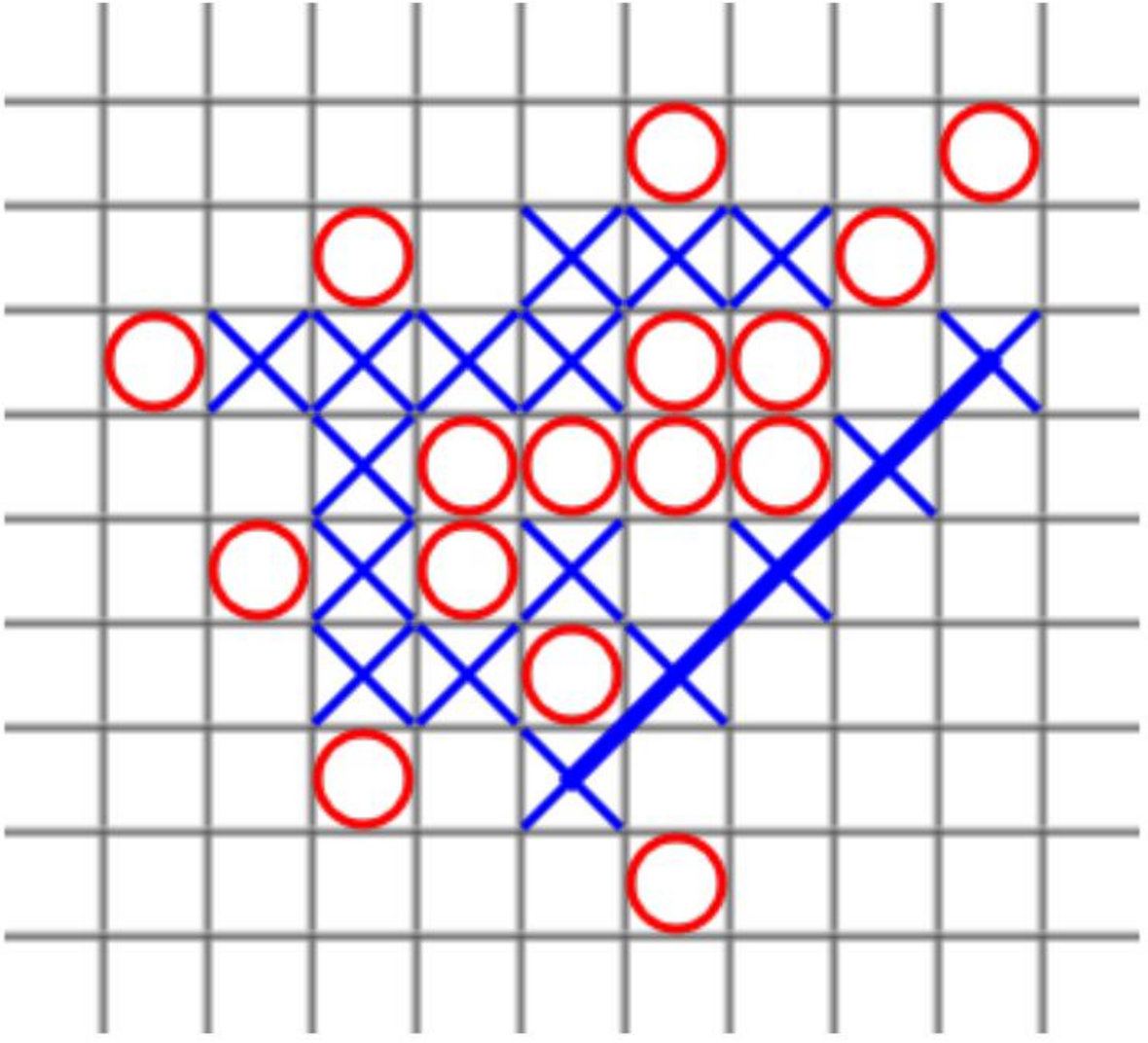
- potřebujeme počítat vše?
- stačí první případ společených narozenin
- algoritmus!

Líné řešení je líné



- potřebujeme počítat vše?
- stačí první případ společených narozenin
- algoritmus!

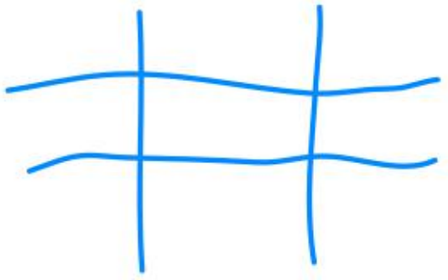
Piškvorky

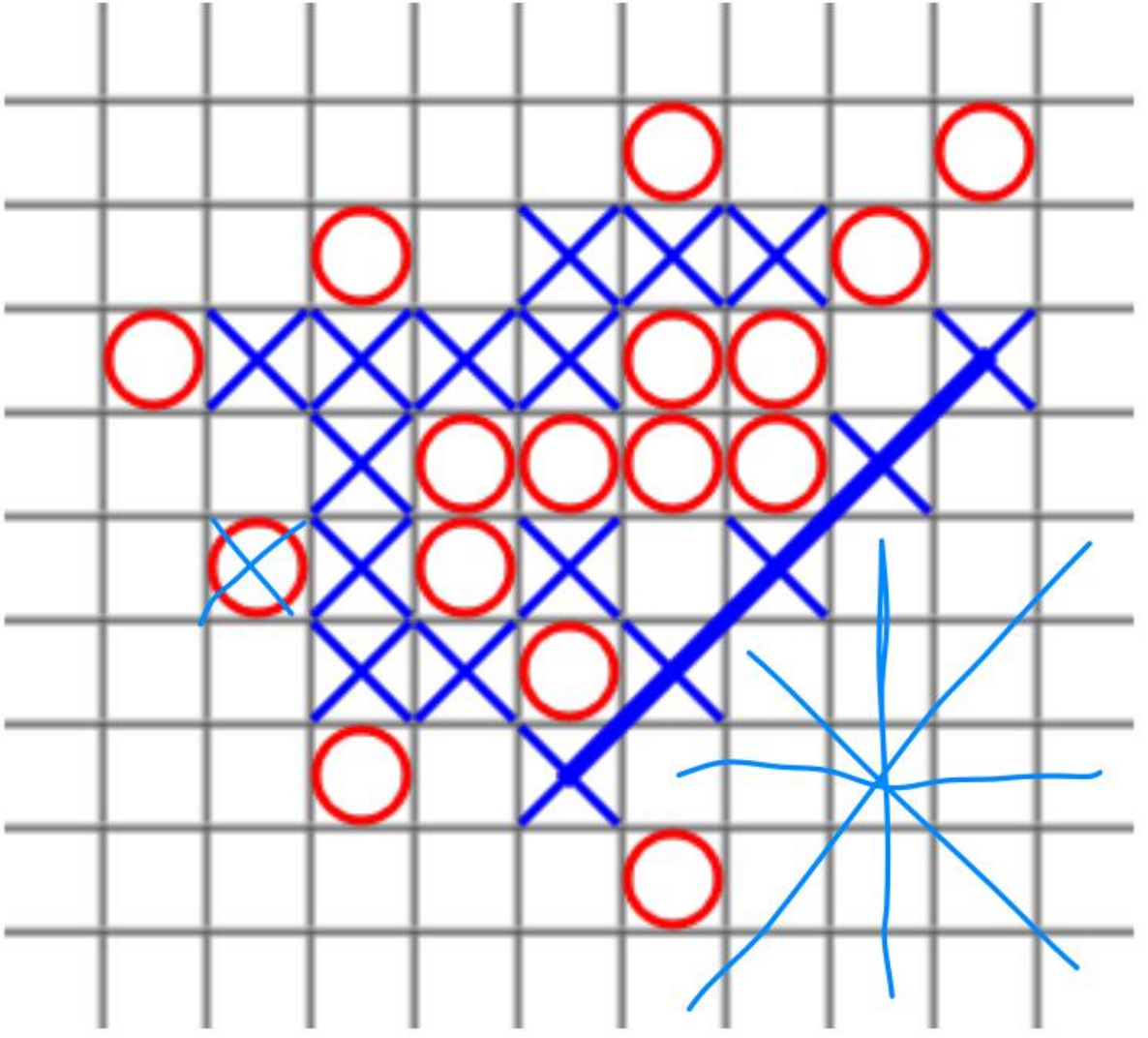


https://cs.wikipedia.org/wiki/Piškvorky

Piškvorky

tic-tac-toe





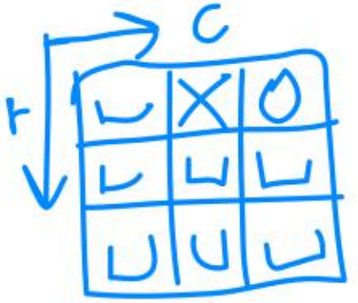
https://cs.wikipedia.org/wiki/Piškvorky

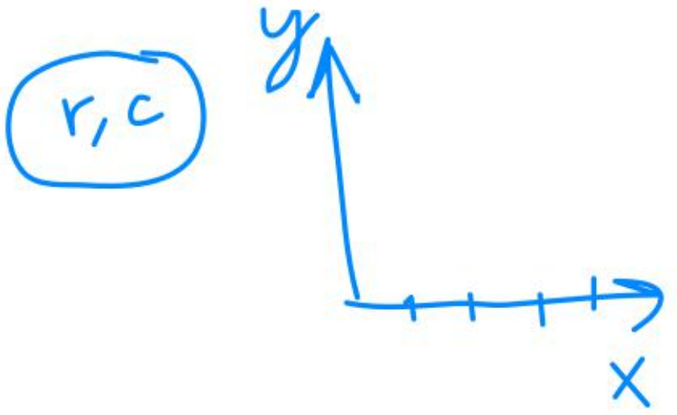
Dekompozice problému

- přemýšlejme jak rozložit složitý problém na jednodušší
- ideálně tak jednoduché, že je triviální je implementovat
- obvykle to pak musíme stejně přepsat
- `my_super_player.play(play_field)`

Dekompozice problému

- přemýšlejme jak rozložit složitý problém na jednodušší
- ideálně tak jednoduché, že je triviální je implementovat
- obvykle to pak musíme stejně přepsat
- `my_super_player.play(play_field)`





Logické funkce

- `is_inside`
- `is_winning`
- `is_empty`
- `is_full`

- Vracejí `True` nebo `False`

- Zpřehledňují hlavní ideu algoritmu

- Vyplatí se i když triviální. Volání funkce něco nestojí …

- ale obvykle program zpomalují jiné věci než volání funkcí

Logické funkce

- `is_inside`
- `is_winning`
- `is_empty`
- `is_full`

- Vracejí `True` nebo `False`

- Zpřehledňují hlavní ideu algoritmu

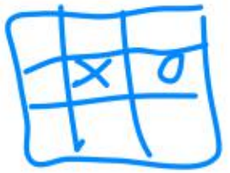
- Vyplatí se i když triviální. Volání funkce něco nestojí …

- ale obvykle program zpomalují jiné věci než volání funkcí

```python
class BasePlayer:
    def __init__(self, mine_sym, opponent_sym, empty_sym):
        self.m = mine_sym
        self.o = opponent_sym
        self.empty = empty_sym
        self.pf = playfield.PlayField(empty_sym=self.empty)

    def play(self, field):
        self.pf.update(field)
        poss_moves = self.pf.get_all_possible_moves()
        return self.find_best_move(poss_moves)

    def find_best_move(self, moves):
        return moves[0]

class RandomPlayer(BasePlayer):
    def find_best_move(self, moves):
        return random.choice(moves)
```

```
1  class BasePlayer:
2      def __init__(self, mine_sym, opponent_sym, empty_sym):
3          self.m = mine_sym
4          self.o = opponent_sym
5          self.empty = empty_sym
6          self.pf = playfield.PlayField(empty_sym=self.empty)
7
8      def play(self, field):
9          self.pf.update(field)
10         poss_moves = self.pf.get_all_possible_moves()
11         return self.find_best_move(poss_moves)
12
13     def find_best_move(self, moves):
14         return moves[0]
15
16 class RandomPlayer(BasePlayer):
17     def find_best_move(self, moves):
18         return random.choice(moves)
```

'X'
'O'

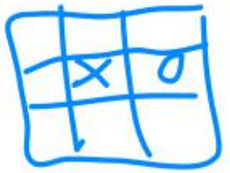


raise NotImplementedError

```
    def get_all_possible_moves(self):
        """
        :return: list of all possible moves (tuples)
        """
        return list(self.empty_pos())

    def empty_pos(self):
        """generate all empty positions"""
        for r, c in self.all_pos():
            if self.is_empty(r, c):
                yield r, c
```

```
1  class BasePlayer:
2      def __init__(self, mine_sym, opponent_sym, empty_sym):
3          self.m = mine_sym
4          self.o = opponent_sym
5          self.empty = empty_sym
6          self.pf = playfield.PlayField(empty_sym=self.empty)
7
8      def play(self, field):
9          self.pf.update(field)
10         poss_moves = self.pf.get_all_possible_moves()
11         return self.find_best_move(poss_moves)
12
13     def find_best_move(self, moves):
14         return moves[0]
15
16 class RandomPlayer(BasePlayer):
17     def find_best_move(self, moves):
18         return random.choice(moves)
```

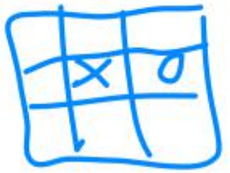
'X'
'O'



raise NotImplementedError

```
1  class BasePlayer:
2      def __init__(self, mine_sym, opponent_sym, empty_sym):
3          self.m = mine_sym
4          self.o = opponent_sym
5          self.empty = empty_sym
6          self.pf = playfield.PlayField(empty_sym=self.empty)
7
8      def play(self, field):
9          self.pf.update(field)
10         poss_moves = self.pf.get_all_possible_moves()
11         return self.find_best_move(poss_moves)
12
13     def find_best_move(self, moves):
14         return moves[0]
15
16 class RandomPlayer(BasePlayer):
17     def find_best_move(self, moves):
18         return random.choice(moves)
```

'X'
'O'



raise NotImplementedError

```
    def get_all_possible_moves(self):
        """
        :return: list of all possible moves (tuples)
        """
        return list(self.empty_pos())

    def empty_pos(self):
        """generate all empty positions"""
        for r, c in self.all_pos():
            if self.is_empty(r, c):
                yield r, c
```

```
1    def get_all_possible_moves(self):
2        """
3            :return: list of all possible moves (tuples)
4        """
5        return list(self.empty_pos())
6
7    def empty_pos(self):
8        """generate all empty positions"""
9        for r, c in self.all_pos():
10           if self.is_empty(r, c):
11               yield r, c
```

self.data ~ 

generátory

- efektivní způsob jak generovat sekvence pro for smyčky
- lepší než: 1) vytvoř seznam, 2) iteruj přes něj
- zpřehlednění programu

```
 1 def generate_squares(max_square):
 2     """
 3     List of squares, starts at 0, stops before > max_square
 4     :param max_square:
 5     :yield:
 6     >>> list(generate_squares(10))
 7     [0, 1, 4, 9]
 8     """
 9     sqr = 0
10     i = 0
11     while sqr<max_square:
12         yield sqr
13         i = i+1
14         sqr = i**2
```

```
>>> [x**2 for x in range(10)]                    generators.py
```

```
 1 def generate_squares(max_square):
 2     """
 3     List of squares, starts at 0, stops before > max_square
 4     :param max_square:
 5     :yield:
 6     >>> list(generate_squares(10))
 7     [0, 1, 4, 9]
 8     """
 9     sqr = 0
10     i = 0
11     while sqr<max_square:
12         yield sqr
13         i = i+1
14         sqr = i**2
```

100

1,4 16

0,1,4

1,2

1,4

serviruj, poskytni

```
>>> [x**2 for x in range(10)]
```

```
for sq in generate_squares(100):
```

1,4

generators.py

```
1 >>> import generators
2 >>> sqg = generators.generate_squares(100)
3 >>> next(sqg)
4 0
5 >>> next(sqg)
6 1
7 >>> next(sqg)
8 4
9 >>> list(sqg)
```

```
1 def generate_squares(max_square):
2     sqr = 0
3     i = 0
4     while sqr<max_square:
5         yield sqr
6         i = i+1
7         sqr = i**2
```

Uvidíme:

A: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

B: [9, 16, 25, 36, 49, 64, 81]

C: chyba za běhu programu

```
>>> import generators
>>> sqg = generators.generate_squares(100)
>>> next(sqg)
0
>>> next(sqg)
1
>>> next(sqg)
4
>>> list(sqg)
```

```
def generate_squares(max_square):
    sqr = 0
    i = 0
    while sqr<max_square:
        yield sqr
        i = i+1
        sqr = i**2
```

Uvidíme:

A: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

B: [9, 16, 25, 36, 49, 64, 81]

C: chyba za běhu programu

```
 1 >>> import generators
 2 >>> sqg = generators.generate_squares(100)
 3 >>> next(sqg)
 4 0
 5 >>> next(sqg)
 6 1
 7 >>> next(sqg)
 8 4
 9 >>> a = list(sqg)
10 >>> next(sqg)
```

```
1 def generate_squares(max_square):
2     sqr = 0
3     i = 0
4     while sqr<max_square:
5         yield sqr
6         i = i+1
7         sqr = i**2
```

Řádek 10 ukáže:

A: 9

B: 0

C: chyba za běhu programu

D: 81

```
 1 >>> import generators
 2 >>> sqg = generators.generate_squares(100)
 3 >>> next(sqg)
 4 0
 5 >>> next(sqg)
 6 1
 7 >>> next(sqg)
 8 4
 9 >>> a = list(sqg)
10 >>> next(sqg)
```

```
1 def generate_squares(max_square):
2     sqr = 0
3     i = 0
4     while sqr<max_square:
5         yield sqr
6         i = i+1
7         sqr = i**2
```

Řádek 10 ukáže:

A: 9

B: 0

C: chyba za běhu programu

D: 81

yield, generátory - proč?

```
1 def is_winning_too_generous(self):
2     for r in range(self.size):
3         for c in range(self.size):
4             if self.is_empty(r,c):
5                 continue
6             for direction in self.directions[0:4]:
7                 if self.is_dir_winning(r,c,direction):
8                     return True
9     return False
```

yield, generátory - proč?

```
1 def is_winning_too_generous(self):
2     for r in range(self.size):
3         for c in range(self.size):
4             if self.is_empty(r,c):
5                 continue
6             for direction in self.directions[0:4]:
7                 if self.is_dir_winning(r,c,direction):
8                     return True
9     return False
```

yield, generátory - proč?

for r,c in all_pos

```
1 def is_winning_too_generous(self):
2     for r in range(self.size):
3         for c in range(self.size):
4             if self.is_empty(r,c):
5                 continue
6             for direction in self.directions[0:4]:
7                 if self.is_dir_winning(r,c,direction):
8                     return True
9     return False
```

pro všechny neprázdné pozice

```
1 def is_winning_too_generous(self):
2     for r,c in self.non_empty_pos():
3         for direction in self.directions[0:4]:
4             if self.is_dir_winning(r,c,direction):
5                 return True
6     return False
```

pro všechny neprázdné pozice

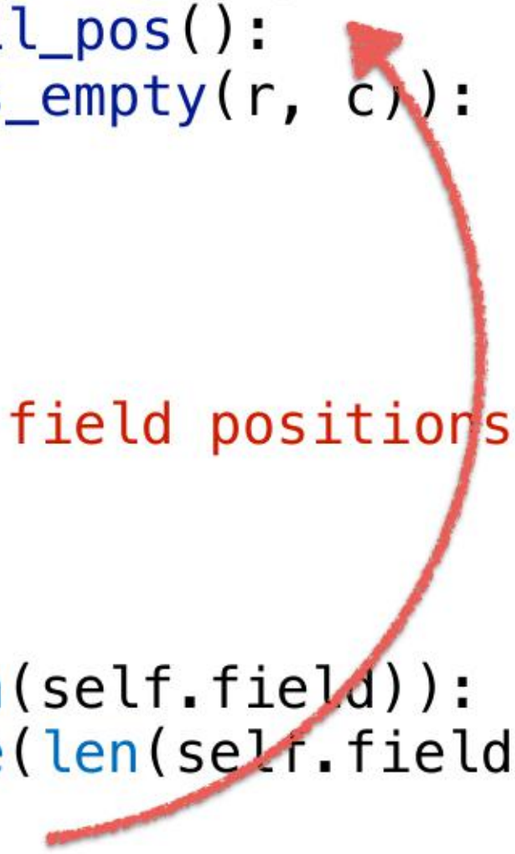
```
1 def is_winning_too_generous(self):
2     for r,c in self.non_empty_pos():
3         for direction in self.directions[0:4]:
4             if self.is_dir_winning(r,c,direction):
5                 return True
6     return False
```

yield místo return

```
1  def non_empty_pos(self):
2      """generate all non-empty positions"""
3      for r,c in self.all_pos():
4          if not(self.is_empty(r, c)):
5              yield r,c
6
7  def all_pos(self):
8      """
9      generator for all field positions
10     :param self:
11     :yield: r,c
12     """
13     for r in range(len(self.field)):
14         for c in range(len(self.field[0])):
15             yield r,c
```

yield místo return

```
1  def non_empty_pos(self):
2      """generate all non-empty positions"""
3      for r,c in self.all_pos():
4          if not(self.is_empty(r, c)):
5              yield r,c
6
7  def all_pos(self):
8      """
9      generator for all field positions
10     :param self:
11     :yield: r,c
12     """
13     for r in range(len(self.field)):
14         for c in range(len(self.field[0])):
15             yield r,c
```

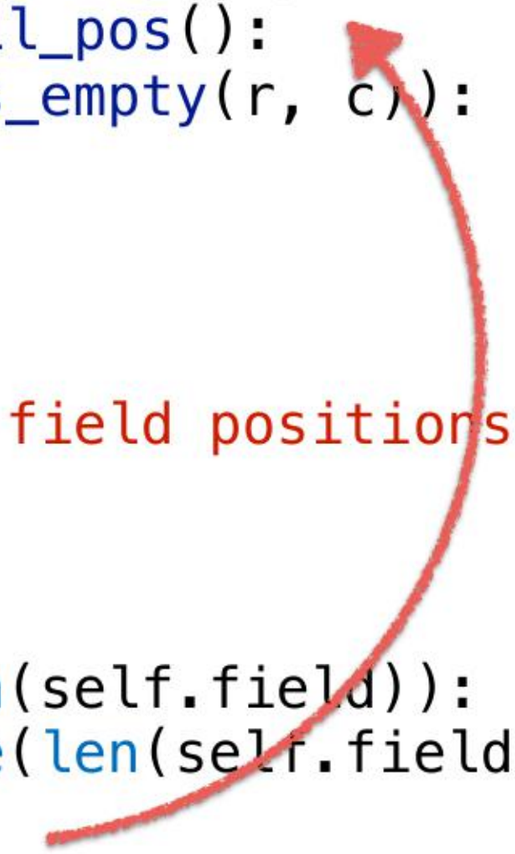


`range` is not a generator

- Často ho tak používáme
- `<class 'range'>`
- Zkusme si ho představit jako *líný* seznam (lazy list)
- má délku, můžeme indexovat

yield místo return

```
1  def non_empty_pos(self):
2      """generate all non-empty positions"""
3      for r,c in self.all_pos():
4          if not(self.is_empty(r, c)):
5              yield r,c
6
7  def all_pos(self):
8      """
9      generator for all field positions
10     :param self:
11     :yield: r,c
12     """
13     for r in range(len(self.field)):
14         for c in range(len(self.field[0])):
15             yield r,c
```



yield místo return

```
1  def non_empty_pos(self):
2      """generate all non-empty positions"""
3      for r,c in self.all_pos():
4          if not(self.is_empty(r, c)):
5              yield r,c
6
7  def all_pos(self):
8      """
9      generator for all field positions
10     :param self:
11     :yield: r,c
12     """
13     for r in range(len(self.field)):
14         for c in range(len(self.field[0])):
15             yield r,c
```