Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана

(национальный исследовательский университет)»

(МГТУ им. Н.Э. Баумана)

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Домашнее задание

по дисциплине «Базовые компоненты интернет-технологий»

Выполнила:

Студентка группы ИУ5-35Б

Ищенко А.С.

2021 г.

Цель домашнего задания: изучение возможностей создания ботов в Telegram и их тестирования.

Задание:

1. Разработайте бота для Telegram. Бот должен реализовывать конечный автомат из трех состояний.
2. Модифицируйте код лабораторной работы №6 таким образом, чтобы он был пригоден для модульного тестирования.
3. Используя материалы лабораторной работы №4 создайте модульные тесты с применением TDD - фреймворка (2 теста) и BDD - фреймворка (2 теста).

Текст программы:

main.py

```
import os
import telebot
from telebot import types
from game import Game
import random

# Токент бота
TOKEN = '5095553179:AAHkhDObVrZH7LWnnZ9x10yMny35xc_y1g8'

# Создание бота
bot = telebot.TeleBot(TOKEN)

game = None

cur_path = os.path.dirname(os.path.abspath(__file__))

@bot.message_handler(commands=['start'])
def cmd_start(message):
   bot.send_message(message.chat.id, 'Добро пожаловать в наше
заведение!')
   markup_inline = types.InlineKeyboardMarkup()
   button_yes = types.InlineKeyboardButton(text='Согласиться',
callback_data="Agree")
   button_no = types.InlineKeyboardButton(text='Отказаться',
callback_data="Disagree")
   markup_inline.add(button_yes, button_no)
   bot.send_message(message.chat.id, 'Сыграете в блэкджек?',
reply_markup=markup_inline)

@bot.callback_query_handler(func=lambda c: c.data == 'Agree')
def handle_agree(callback_query: types.CallbackQuery):
   global game
   if game is not None:
       bot.send_message(callback_query.from_user.id, 'Игра уже
начата')
       return
   game = Game()

   bot.send_message(callback_query.from_user.id, 'Ваши карты:' + "\n"
+ game.client.print_cards())
   bot.send_message(callback_query.from_user.id, "Очки: " +
str(game.client.sum()))
   winner = check_finish()
   if winner is not None:
       congratulation_message(callback_query, winner)
       return

   bot.send_message(callback_query.from_user.id, 'Карта бота:' + "\n"
+ game.bot.print_cards(first_card=True))
   # bot.send_message(callback_query.from_user.id,  "Bot " +
str(game.bot.sum()))
   winner = check_finish()
   if winner is not None:
```

```
        congratulation_message(callback_query, winner)
        return

    else:
        markup_inline = send_question()
        bot.send_message(callback_query.from_user.id, 'Еще?)',
reply_markup=markup_inline)

@bot.callback_query_handler(func=lambda c: c.data == 'Disagree')
def handle_disagree(callback_query: types.CallbackQuery):
    finish(callback_query)

@bot.callback_query_handler(func=lambda c: c.data == 'No')
def handle_no(callback_query: types.CallbackQuery):
    if game is None:
        return
    winner = check_finish()

    if winner is None:
        if game.bot.sum() > game.client.sum():
            message = 'Победитель бот :('
        elif game.bot.sum() < game.client.sum():
            message = 'Вы победитель :)'
        elif game.bot.sum() == game.client.sum():
            message = 'Ничья :|'
        bot.send_message(callback_query.from_user.id, message)
        open_the_cards(callback_query)
        finish(callback_query)
        return
    congratulation_message(callback_query, check_finish())
    return

@bot.callback_query_handler(func=lambda c: c.data == 'Yes')
def handle_yes(callback_query: types.CallbackQuery):
    if five_cards(callback_query): return

    add_cards_to_client()
    bot.send_message(callback_query.from_user.id, 'Ваши карты:' + "\n"
+ game.client.print_cards())
    bot.send_message(callback_query.from_user.id, "Очки: " +
str(game.client.sum()))
    winner = check_finish()
    if winner is not None:
        congratulation_message(callback_query, winner)
        return

    add_cards_to_bot()
    # bot.send_message(callback_query.from_user.id,  "Bot " +
str(game.bot.sum()))
    winner = check_finish()
    if winner is not None:
        congratulation_message(callback_query, winner)
        return

    markup_inline = send_question()
    bot.send_message(callback_query.from_user.id, 'Еще?)',
reply_markup=markup_inline)
```

```
def send_question():
   markup_inline = types.InlineKeyboardMarkup()
   button_yes = types.InlineKeyboardButton(text='Да',
callback_data="Yes")
   button_no = types.InlineKeyboardButton(text='Нет',
callback_data="No")
   markup_inline.add(button_yes, button_no)
   return markup_inline

# доделать
def check_finish():
   bot_sum = game.bot.sum()
   client_sum = game.client.sum()

   if client_sum == 21:
       return 'client'
   elif bot_sum > 21:
       return 'client'
   elif bot_sum == 21:
       return 'bot'
   elif client_sum > 21:
       return 'bot'

   return None


def add_cards_to_client():
   game.add_new_cards_to_client()

def add_cards_to_bot():
   game.add_new_cards_to_bot()

def five_cards(callback_query):
   if game.client.count_cards() >= 5:
       bot.send_message(callback_query.from_user.id, 'У Вас 5 карт!
Вскрываемся!')
       congratulation_message(callback_query, check_finish())
       return True

def congratulation_message(callback_query, winner):
   bot.send_message(callback_query.from_user.id, 'Победитель ' +
winner + ' !')
   open_the_cards(callback_query)
   finish(callback_query)
   return

def open_the_cards(callback_query):
   bot.send_message(callback_query.from_user.id,
'____________________________')
   bot.send_message(callback_query.from_user.id, '\nИТОГИ:\n')
   bot.send_message(callback_query.from_user.id, 'Ваши карты:' + "\n"
+ game.client.print_cards())
   bot.send_message(callback_query.from_user.id, "Очки: " +
str(game.client.sum()))
   bot.send_message(callback_query.from_user.id, '\nКарты бота:' +
"\n" + game.bot.print_cards())
```

```
    bot.send_message(callback_query.from_user.id, "Очки: " +
str(game.bot.sum()))
    bot.send_message(callback_query.from_user.id,
'____________________________')

def finish(callback_query):
    global game
    game = None
    bot.send_message(callback_query.from_user.id, 'До свидания!\nБудем
рады видеть Вас снова')

if __name__ == '__main__':
    bot.infinity_polling()
```

create_cards.py

```
from card import Card

clubs_suit = 'Пики'
diamonds_suit = 'Буби'
hearts_suit = 'Черви'
spades_suit = 'Кресты'

suits = [clubs_suit, diamonds_suit, hearts_suit, spades_suit]

a_number = 'Туз'
second_number = '2'
third_number = '3'
fourth_number = '4'
fifth_number = '5'
sixth_number = '6'
seventh_number = '7'
eleventh_number = '8'
nineth_number = '9'
tenth_number = '10'
king_number = 'Король'
lady_number = 'Дама'
jack_number = 'Валет'

numbers = [a_number,
           second_number,
           third_number,
           fourth_number,
           fifth_number,
           sixth_number,
           seventh_number,
           eleventh_number,
           nineth_number,
           tenth_number,
           king_number,
           lady_number,
           jack_number]

weights = {
   a_number: 11,
   second_number: 2,
   third_number: 3,
   fourth_number: 4,
   fifth_number: 5,
   sixth_number: 6,
   seventh_number: 7,
   eleventh_number: 8,
   nineth_number: 9,
   tenth_number: 10,
   king_number: 6,
   lady_number: 4,
   jack_number: 2
}

def create_cards():
   cards = {}
```

```
for suit in suits:
    for number in numbers:
        card = Card(suit, number, weights[number])
        cards[card.id] = card
return cards
```

card.py

```
class Card:
   def __init__(self, suit, number, weight):
       self.id = suit + "__" + number
       self.suit = suit
       self.number = number
       self.weight = weight
       if self.suit == "Clubs":
           self.directory = "cards/" + self.suit + "/" + self.number +
".png"

   def print_card_name(self):
       return self.suit + " " + self.number

   # def dir_to_card_picture(self):
   #     if self.suit == "Clubs":
   #         directory = "cards/" + self.suit + "/" + self.number +
".JPG"
   #         return directory
```

gamer.py

```
MAX_CARD_COUNT = 5

class Gamer:
    def __init__(self):
        self.cards = []

    def print_cards(self, **args):
        string = ''
        number = 0
        mode = args.get('first_card')
        if mode:
            string += "1: " + self.cards[0].print_card_name() + "\n"
            return string
        for card in self.cards:
            number += 1
            string += str(number) + ": " + card.print_card_name() +
"\n"
        return string
        # dirs = []
        # for card in self.cards:
        #     dirs.append(card.dir_to_card_picture())
        # return dirs

    def add_new_card(self, card):
        if self.count_cards() < 5:
            self.cards.append(card)

    def sum(self):
        sum = 0
        for card in self.cards:
            sum += card.weight
        return sum

    def count_cards(self):
        return len(self.cards)

#bot = Gamer()
#client = Gamer()

#bot.add_new_card()
#client.add_new_card(Card(clubs_suit, a_number, weights[a_number]))
```

game.py

```python
from gamer import Gamer
from create_cards import create_cards
import random

class Game:
    def __init__(self):
        self.all_cards = create_cards()
        self.bot = Gamer()
        self.client = Gamer()
        self.init_first_cards()

    def get_available_cards(self):
        cards_map = self.all_cards.copy()
        for card in self.bot.cards:
            del cards_map[card.id]
        for card in self.client.cards:
            del cards_map[card.id]
        return cards_map


    def init_first_cards(self):
        for i in range(0, 2):
            available_cards = self.get_available_cards()
            available_cards_keys = list(available_cards.keys())
            rand_id = random.choice(available_cards_keys)
            self.bot.add_new_card(available_cards[rand_id])

        for i in range(0, 2):
            available_cards = self.get_available_cards()
            available_cards_keys = list(available_cards.keys())
            rand_id = random.choice(available_cards_keys)
            self.client.add_new_card(available_cards[rand_id])

    def add_new_cards_to_bot(self):
        available_cards = self.get_available_cards()
        available_cards_keys = list(available_cards.keys())
        rand_id = random.choice(available_cards_keys)
        self.bot.add_new_card(available_cards[rand_id])

    def add_new_cards_to_client(self):
        available_cards = self.get_available_cards()
        available_cards_keys = list(available_cards.keys())
        rand_id = random.choice(available_cards_keys)
        self.client.add_new_card(available_cards[rand_id])
```

TDD:

tdd.py

```
import unittest
from gamer import Gamer
from card import Card
from tdd.check_for_testing import check_finish


class TestStreetsAndHouses(unittest.TestCase):

    def test_add_new_card(self):
        gamer_for_tests = Gamer()
        gamer_for_tests.add_new_card(Card("Clubs", "king", 6))
        expected_result = "1: Clubs king\n"
        self.assertEqual(expected_result,gamer_for_tests.print_cards())

    def test_count_sum(self):
        gamer_for_tests = Gamer()
        gamer_for_tests.add_new_card(Card("Clubs", "king", 6))
        gamer_for_tests.add_new_card(Card("Diamonds", "king", 6))
        gamer_for_tests.add_new_card(Card("Hearts", "king", 6))
        expected_result = 18
        self.assertEqual(expected_result, gamer_for_tests.sum())


    def test_check_finish_with_winner(self):
        player1 = Gamer()
        player2 = Gamer()

        player1.add_new_card(Card("Clubs", "a", 11))
        player1.add_new_card(Card("Hearts", "10", 10))

        player2.add_new_card(Card("Hearts", "2", 2))
        player2.add_new_card(Card("Diamonds", "king", 6))
        expected_result = 'player1'
        self.assertEqual(expected_result, check_finish(player1,
player2))


    def test_check_finish_no_winner(self):
        player1 = Gamer()
        player2 = Gamer()

        player1.add_new_card(Card("Clubs", "2", 2))
        player1.add_new_card(Card("Hearts", "10", 10))

        player2.add_new_card(Card("Hearts", "2", 2))
        player2.add_new_card(Card("Diamonds", "king", 6))
        expected_result = None
        self.assertEqual(expected_result, check_finish(player1,
player2))


if __name__ == "__main__":
    unittest.main()
```

BDD:

bdd_tests_feature.py

```
Feature: get_roots function

 Scenario: test no winner
     Given player1 cards: 2 clubs, 10 hearts; player2 cards: 2 hearts,
king diamonds
     When check_finish run
     Then there is no winner

 Scenario: test any winner
    Given player1 cards: a clubs, 10 hearts; player2 cards: 2 hearts,
king diamonds
    When check_finish run_
    Then winner is player1

 Scenario: test new card
    Given gamer has no cards
    When gamer gets clubs king
    Then gamers cards are: clubs king

 Scenario: test count sum
    Given gamer has no cards_
    When gamer gets clubs king, diamonds king, hearts king
    Then the sum is 18
```

test_no_winner.py

```
from behave import *
from gamer import Gamer
from card import Card
from tdd.check_for_testing import check_finish

@given('player1 cards: 2 clubs, 10 hearts; player2 cards: 2 hearts,
king diamonds')
def step_impl(context):
   context.player1 = Gamer()
   context.player2 = Gamer()

   context.player1.add_new_card(Card("Clubs", "2", 2))
   context.player1.add_new_card(Card("Hearts", "10", 10))

   context.player2.add_new_card(Card("Hearts", "2", 2))
   context.player2.add_new_card(Card("Diamonds", "king", 6))
   pass

@when('check_finish run')
def step_impl(context):
   context.winner = check_finish(context.player1, context.player2)
   pass

@then('there is no winner')
def step_impl(context):
   assert context.winner == None
```

test_any_winner.py

```
from behave import *
from gamer import Gamer
from card import Card
from tdd.check_for_testing import check_finish

@given('player1 cards: a clubs, 10 hearts; player2 cards: 2 hearts,
king diamonds')
def step_impl(context):
   context.player1 = Gamer()
   context.player2 = Gamer()

   context.player1.add_new_card(Card("Clubs", "a", 11))
   context.player1.add_new_card(Card("Hearts", "10", 10))

   context.player2.add_new_card(Card("Hearts", "2", 2))
   context.player2.add_new_card(Card("Diamonds", "king", 6))
   expected_result = 'player1'
   pass

@when('check_finish run_')
def step_impl(context):
   context.winner = check_finish(context.player1, context.player2)
   pass

@then('winner is player1')
def step_impl(context):
   assert context.winner == "player1"
```

test_new_card.py

```
from behave import *
from gamer import Gamer
from card import Card

@given('gamer has no cards')
def step_impl(context):
   context.gamer = Gamer()
   pass

@when('gamer gets clubs king')
def step_impl(context):
   context.gamer.add_new_card(Card("Clubs", "king", 6))
   pass

@then('gamers cards are: clubs king')
def step_impl(context):
   assert context.gamer.print_cards() == "1: Clubs king\n"
```

test_count_sum.py

```
from behave import *
from gamer import Gamer
from card import Card

@given('gamer has no cards_')
def step_impl(context):
   context.gamer = Gamer()
   pass

@when('gamer gets clubs king, diamonds king, hearts king')
def step_impl(context):
   context.gamer.add_new_card(Card("Clubs", "king", 6))
   context.gamer.add_new_card(Card("Diamonds", "king", 6))
   context.gamer.add_new_card(Card("Hearts", "king", 6))
   pass

@then('the sum is 18')
def step_impl(context):
   assert context.gamer.sum() == 18
```

check_for_testing.py

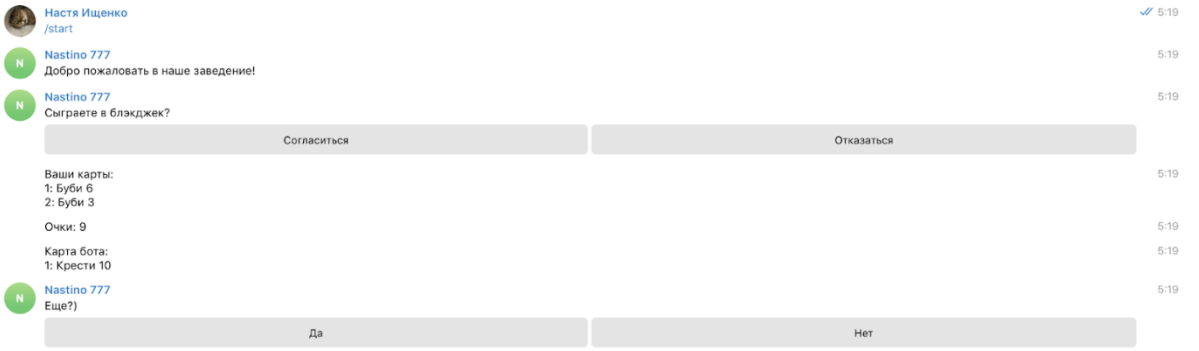
```
def check_finish(player1, player2):
    player1_sum = player1.sum()
    player2_sum = player2.sum()

    if player2_sum == 21:
        return 'player2'
    elif player1_sum > 21:
        return 'player2'
    elif player1_sum == 21:
        return 'player1'
    elif player2_sum > 21:
        return 'player1'

    return None
```

Результат выполнения программы:

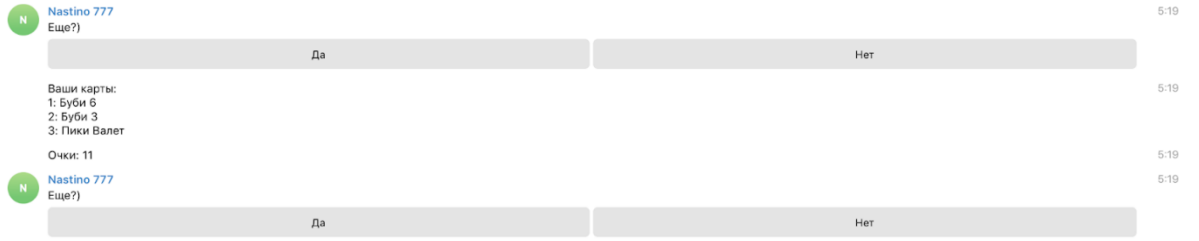
main.py

Нажатие кнопки “Отказаться”

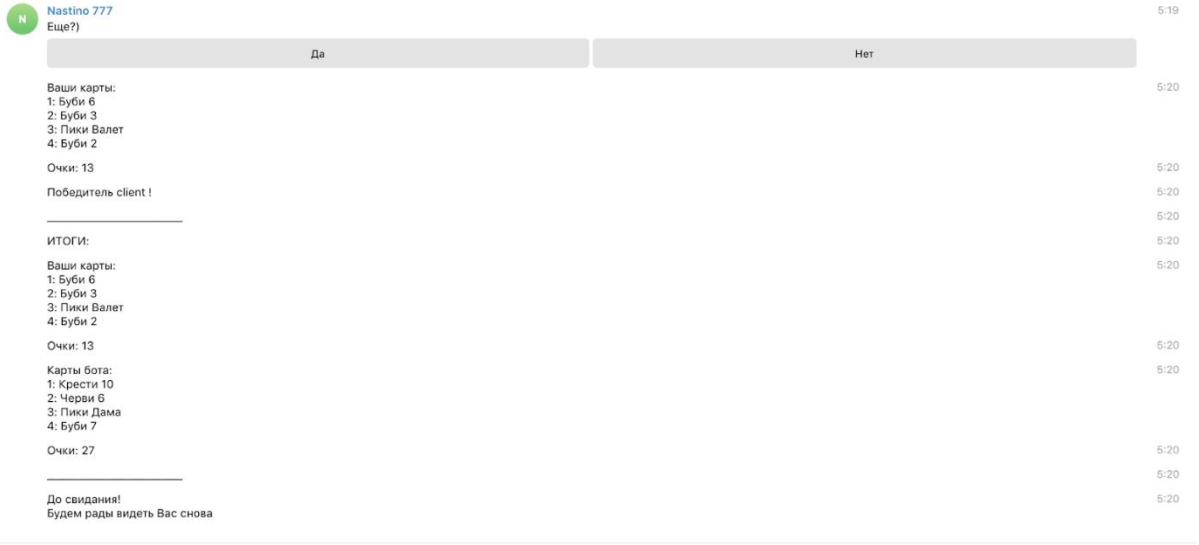
Нажатие кнопки “Согласиться”



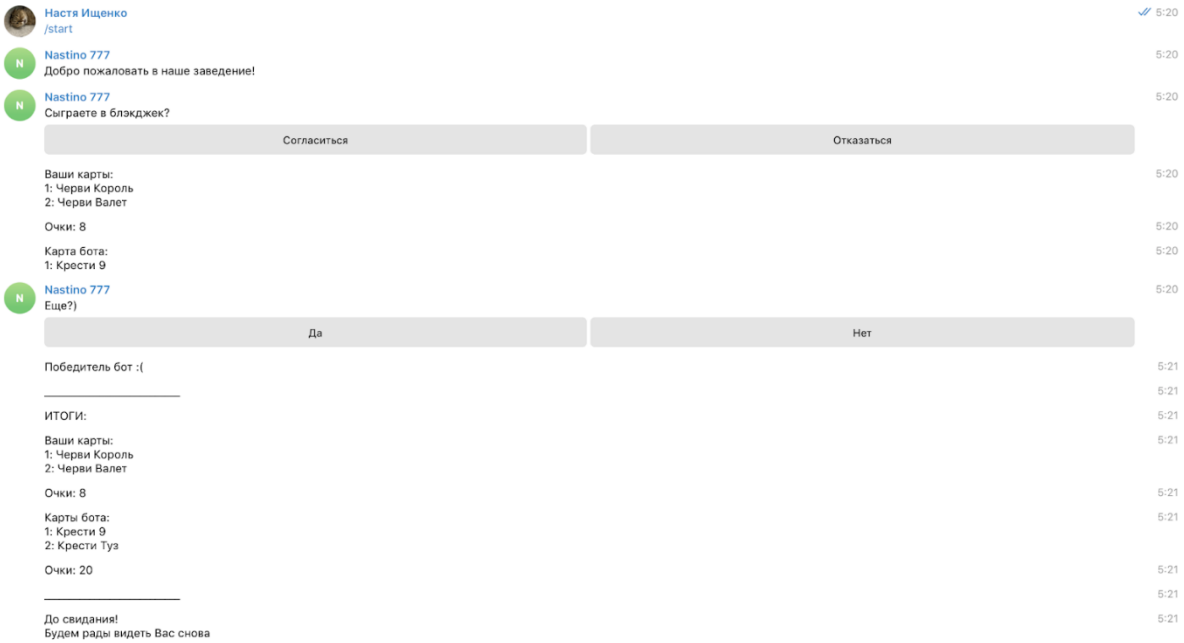
Нажатие кнопки “Да”



Повторное нажатие кнопки “Да”



Нажатие кнопки “Нет”



TDD
tdd.py

```
Testing started at 5:22 ...
Launching unittests with arguments python -m unittest



Ran 4 tests in 0.003s

OK

Process finished with exit code 0
```

BDD

```
Feature: get_roots function # features/bdd_tests.feature:1

  Scenario: test no winner                                                    # features/bdd_tests.feature:3
    Given player1 cards: 2 clubs, 10 hearts; player2 cards: 2 hearts, king diamonds # features/steps/test_no_winner.py:6 0.000s
    When check_finish run                                                     # features/steps/test_no_winner.py:18 0.000s
    Then there is no winner                                                   # features/steps/test_no_winner.py:23 0.000s

  Scenario: test any winner                                                   # features/bdd_tests.feature:8
    Given player1 cards: a clubs, 10 hearts; player2 cards: 2 hearts, king diamonds # features/steps/test_any_winner.py:6 0.000s
    When check_finish run_                                                    # features/steps/test_any_winner.py:19 0.000s
    Then winner is player1                                                    # features/steps/test_any_winner.py:24 0.000s

  Scenario: test new card             # features/bdd_tests.feature:13
    Given gamer has no cards          # features/steps/test_new_card.py:5 0.000s
    When gamer gets clubs king        # features/steps/test_new_card.py:10 0.000s
    Then gamers cards are: clubs king # features/steps/test_new_card.py:15 0.000s

  Scenario: test count sum                                # features/bdd_tests.feature:18
    Given gamer has no cards_                             # features/steps/test_count_sum.py:5 0.000s
    When gamer gets clubs king, diamonds king, hearts king # features/steps/test_count_sum.py:10 0.000s
    Then the sum is 18                                    # features/steps/test_count_sum.py:17 0.000s

1 feature passed, 0 failed, 0 skipped
4 scenarios passed, 0 failed, 0 skipped
12 steps passed, 0 failed, 0 skipped, 0 undefined
Took 0m0.002s
```