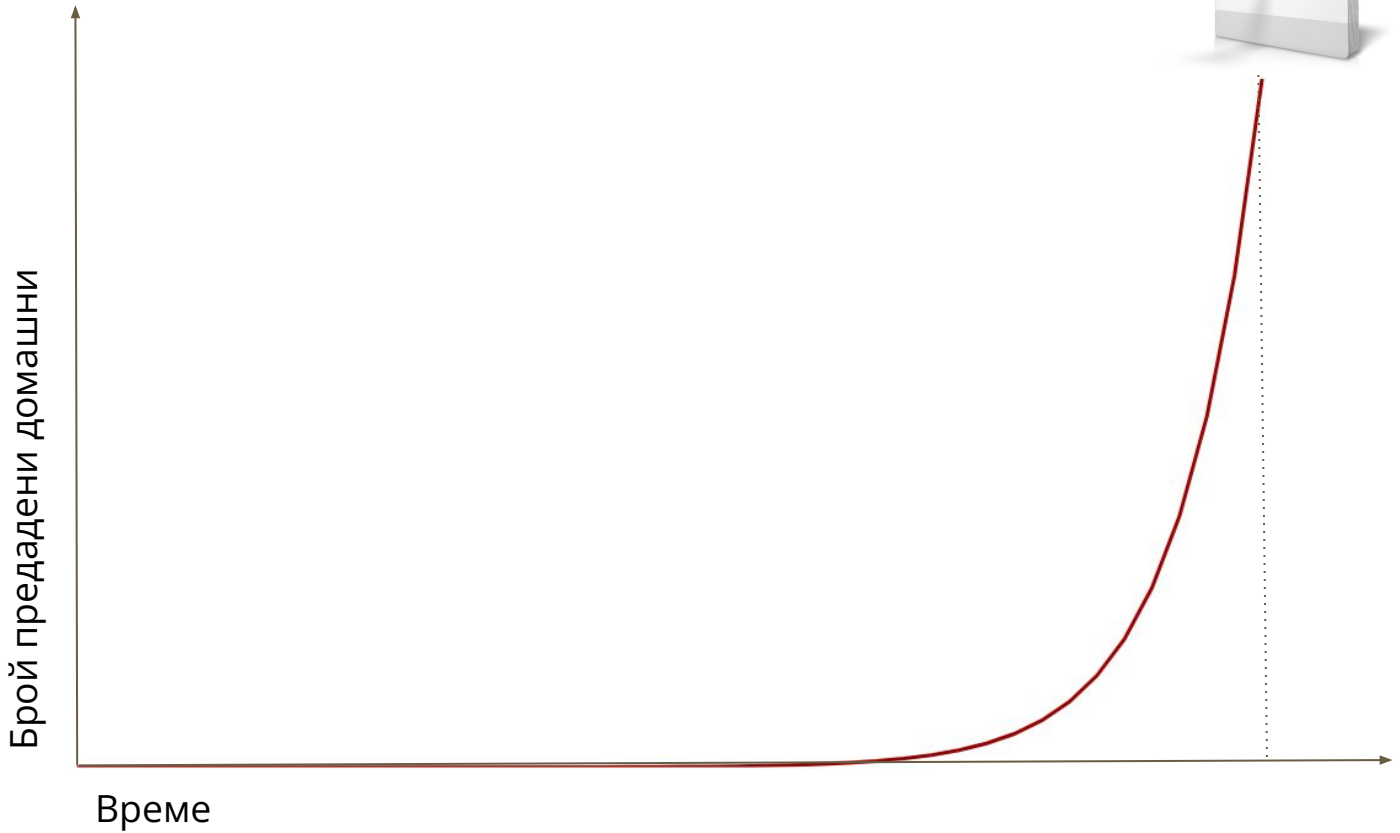
07. Изключения и with

03 ноември 2022

Но първо - какво е това?



Съвети за домашното

- Не оставяйте решението за последния ден:
  - защото може да не се справите;
  - защото нямаме време да ви дадем обратна връзка.
- Винаги имайте предвид празния input.
- Ако ще споделяте тестове, направете ги кадърно:
  - всяка функция (test case) трябва да тества за конкретно нещо, а не да съдържа всевъзможни инпути;
  - избирайте подходящи имена за тестовите методи, такива, които обясняват какво “изискване” към функционалността тества въпросният метод, а не test_1, test_2...
  - същото важи и за класовете, очевидно
  - слагайте Docstring на тестовете си, за да се вижда какво тествате;
    - BTW - слагайте Docstring **и в кода си**, за да се вижда какво правите.
- Четете вече зададените въпроси

Бързо ревю на често срещаните грешки

- Сравнение на числа
- (Не) хубави return-s
- enumerate
- Ненужни проверки
- Тернарен синтаксис
- ><
- Малко абстрактни забележки

Първо - disclaimer

Някои от грешките, които ще обсъдим може да са налични и при вас, но да не сме ви оставили коментар.

След 15 еднакви грешки просто решаваме да го индикираме на лекция, вместо да оставяме 50 еднакви коментара.

Числа се сравняват с ==, а не с is

Case in point:

```
>>> x = 5
>>> y = 5
>>> x == y
True
>>> x is y
True
```

Супер!

Обаче!

```
>>> a = 3**20
>>> b = 3**20
>>> a == b
True
>>> a is b
False
```

Не-супер!

Връщайте (булеви) изрази директно

“Лошо”:

```
if nums_to_angle(text_to_nums(word)) % len(word) == 0:
    return True
else:
    return False
```

“Хубаво”:

```
return nums_to_angle(text_to_nums(word)) % len(word) == 0
```

Сходна тема

Напълно окей е да направите следното:

```
return <operation returns a valid result or false> or 5
```

Нагледно:

```
>>> 5 % 5 or 'baba'
'baba'
>>> 6 % 5 or 'baba'
1
```

enumerate > range

“Лошо”:

```
for i in range(len(nums)):
    if nums[i] % 4 == 0:
      <do something with i>
```

“Хубаво”:

```
for i, num in enumerate(nums):
    if num % 4 == 0:
      <do something with i>
```

Мислете дали проверките ви са нужни

Пример:

```
if angle < 0: # <- излишно
    while angle < 0:
        angle += 360
```

Ползвайте тернарният оператор

“Лошо”:

```
if num % 4 == 0:
    divisor = 4
else:
    divisor = 3
```

“Хубаво”:

```
divisor = 4 if num % 4 == 0 else 3
```

Chain-вайте операторите за сравнение (когато можете)

“Лошо”:

```
if angle > 15 and angle <= 45:
    <do_something>
```

“Хубаво”:

```
if 15 < angle <= 45:
    <do_something>
```

Малко абстрактни забележки

- Внимавайте с подхода if-elif-elif-elif-elif... А ако бяха 100?
- Взимайте предвид празен инпут - празен низ, празна колекция
- Празните редове преди return са доста излишни
- Не слагайте скоби около if условия и return стейтмънти

- Не сме ви “мрънкали” за дължина на редовете?
- Също така и за двата празни реда около top-level дефиниции?
- Второто го имайте предвид за следващото домашно
- Първото го имайте предвид в някакви нормални граници - 100 е окей, да си пишете домашните като Виктор - не

Обратно на темата - Изключения (Exceptions)

- Най-лесно се учи с пример
- Особено, когато примерът се запомня лесно

```
import Wife

my_wife = Wife()

if my_wife.allows_me_to_go_outside():
    me.go_outside()
else:
    me.notify_buddies('Под чехъл съм!')

try:
    me.go_outside()
except MadWifeError:
    me.notify_buddies('Под чехъл съм!')
```

Поискай разрешение vs Моли се за прошка

```
if my_wife.allows_me_to_go_outside():
    me.go_outside()
else:
    me.notify_buddies('Под чехъл съм!')
```

```
try:
    me.go_outside()
except MadWifeError:
    me.notify_buddies('Под чехъл съм!')
```

По тази тема по-късно

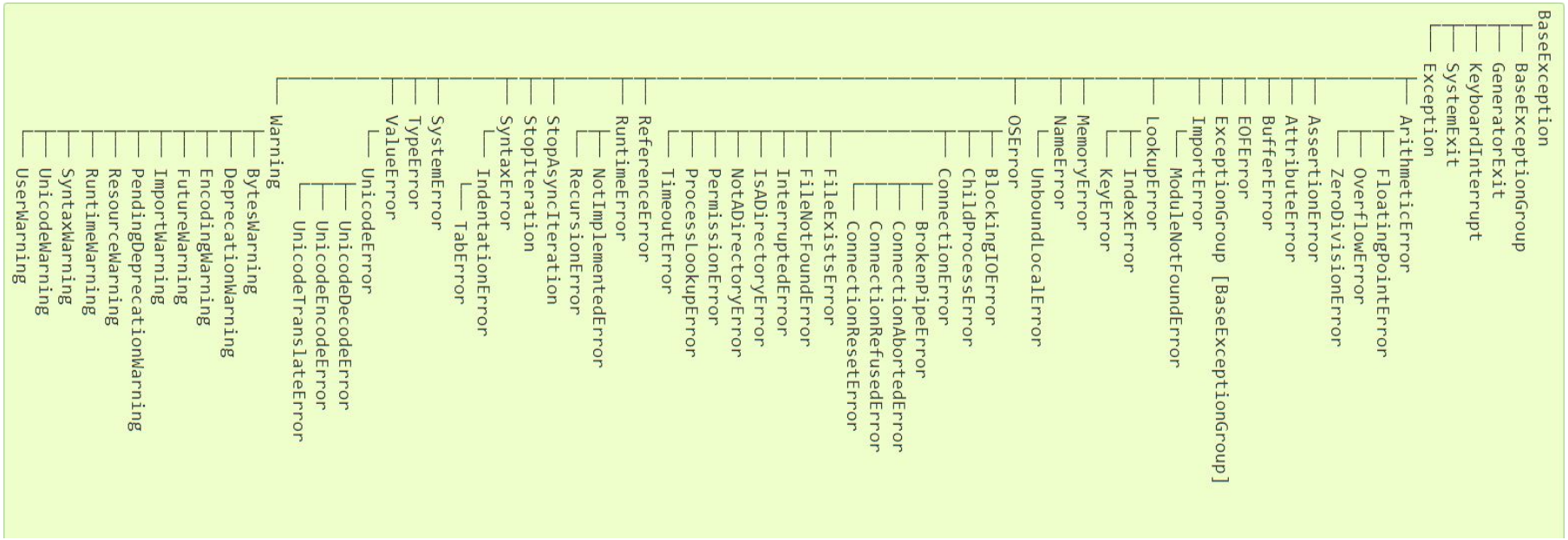
Но какво всъщност представлява try/except?

- Изключението е грешка или неочаквана аномалия, изискваща специална обработка, променяща нормалното протичане на изпълнението на програмата.
- try/except може да “хване” изключението и да се справи с изненадата

```
try:
    # блок
except Изключение, или tuple от Изключения:
    # блок за хващане и обработка на някое от описаните изключения
except ДругоИзключение:
    # блок за хващане и обработка на някое от описаните изключения
…
except:
    # блок за хващане и обработка на което и да е изключение(except BaseException)
else:
    # блок изпълняващ се, ако не е възникнала изключителна ситуация
finally:
    # блок изпълняващ се винаги
```

Стандартни изключения

- Списъкът е голям и не се хваща на слайда, така че...



- Ако искате да ги разгледате, [тук са](#).

Една извадка като за начало

```
something_not_existing
# NameError: name 'something_not_existing' is not defined

Some_dict['unexisting_key']
# KeyError: 'unexisting_key'

1 + 'edno'
# TypeError: unsupported operand type(s) for +: 'int' and 'str'

int('edno')
# ValueError: invalid literal for int() with base 10: 'edno'

Some_list_with_two_elements[3]
# IndexError: list index out of range

for x in collection:
print(x) # IndentationError: expected an indented block after 'for' statement

'Az'.sum()
# AttributeError: 'str' object has no attribute 'sum'
```

Ако искате да използвате изключението, можете...

```
try:
    me.go_outside()
except MadWifeError as data:
    me.notify_buddies(f'Под чехъл съм! Жената каза {data}!')
```

Какво ще има в `data` зависи от самото изключение, но е прието всички да връщат годна за отпечатване стойност, ако се дадат като аргументи на `str` или `repr`.

Ако има няколко изключения, за които искате една и съща реакция

```
try:
    me.go_outside()
except MadWifeError as data:
    me.notify_buddies(f'Под чехъл съм! Жената каза {data}!')
except (CarStarterError, DiarrheaError):
    me.notify_buddies('Пас съм момчета. Технически проблеми')
```

Да допълним примера с останалите опции за try

```
try:
    me.go_outside()
except MadWifeError as data:
    me.notify_buddies(f'Под чехъл съм! Жената каза {data}!')
except (CarStarterError, DiarrheaError):
    me.notify_buddies('Пас съм момчета. Технически проблеми.')
except:
    me.notify_buddies('Пас съм момчета. Не знам защо, но не мога да изляза.')
else:
    me.notify_buddies('Идвам. Режете мезето!')
finally:
    me.drink_beer() # И да паднем, и да бием...
    me.apologize_to_wife() # Тя винаги ще е сърдита
```

А как да дефинирам свое собствено изключение 1/3

Можете просто да наследите `Exception` и това би било достатъчно.

```
class MadWifeError(Exception):
    pass


raise MadWifeError() # MadWifeError
```

А как да дефинирам свое собствено изключение 2/3

Можете да дефинирате собствена грешка, която да се използва по подразбиране.

```
class MadWifeError(Exception):
    """Exception raised by a mad wife."""

    def __init__(self, message='Ядосана съм и ти си знаеш защо.'):
        self._message = message
        super().__init__(self._message)


raise MadWifeError() # MadWifeError: Ядосана съм и ти си знаеш защо.
raise MadWifeError("Вече не ме обичаш!") # MadWifeError: Вече не ме обичаш!
```

А как да дефинирам свое собствено изключение 3/3

Можете да промените начина, по който изключението се евалюира като текст.

```
class MadWifeError(Exception):
        """Exception raised by a mad wife."""

        def __init__(self, message='Ядосана съм и ти си знаеш защо.'):
            self._message = message
            super().__init__(self._message)

        def __str__(self):
            return f'Глупак, простак, мръсник, циник! {self._message}'


raise MadWifeError() # MadWifeError: Глупак, простак, мръсник, циник! Ядосана съм
и ти си знаеш защо.
```

Да искам разрешение, или да се моля за прошка?

Има два основни подхода:

- "EAFP" - "Easier to Ask for Forgiveness than Permission”
- "LBYL" - "Look Before You Leap".

В тази лекция говорим за Изключения, така че няма как да не защитим първия…

EAFP - в повечето случаи е по-бърз

```
if not box.empty():
    box.pick_item()

try:
    box.pick_item()
except:
    <някой да напълни кутията>
```

Ако приемем, че с повечето си опити кутията няма да е празна, try/except ще спести много операции.

EAFP - може да се справи с няколко проблема

```
if not box.empty() and not box.locked():
    box.pick_item()

try:
    box.pick_item()
except BoxEmptyError:
    <някой да напълни кутията>
except BoxLockedError:
    <някой да донесе ключ>
```

EAFP - може да се справи с ”незнайни” проблеми

```
if not box.empty() and not box.locked():
    box.pick_item()

try:
    box.pick_item()
except BoxEmptyError:
    <някой да напълни кутията>
except BoxLockedError:
    <някой да донесе ключ>
except:
    <може би box.empty вече не работи правилно?>
```

Все пак да дадем малко кредит и на LBYL

- EAFP може да има нежелани странични ефекти.

```
if not box.empty() and not box.locked():
    box.pick_item()

try:
    box.pick_item()
except BoxLockedError:
    <прекалено късно - алармата се включи>
```

Добре, а кога да (не) try/except-вам? 1/4

- Когато Python се натъкне на изключение в даден блок и в него то не се обработи, изключението се праща към горния блок, после към по-горния и така, докато изключението не бъде прехванато или не стигнем най-отгоре и интерпретаторът не спре програмата по познатия ни вече начин.
- Не използвайте try/catch просто за да предефинирате грешката и да я пратите нагоре в по-външен блок от кода ви.

```
try:
    me.go_outside()
except MadWifeError:
    raise RuntimeError('Wife is mad')
```

- Най-вероятно оригиналната грешка има повече информация и ще е по-полезна

Добре, а кога да (не) try/except-вам? 2/4

- Не хващайте изключение, с което не можете да се справите и нямате план за действие

```
def homework(text):
    try:
        return text.split()
    except AttributeError:
        return None

print(homework(666)) # None


def homework(text):
    return text.split()

print(homework(666)) # AttributeError: 'int' object has no attribute 'split'
```

Добре, а кога да (не) try/except-вам? 3/4

Един от 100-те съвета, които може да видите в “The Pragmatic Programmer” е “Crash early”

- Ако в програмата ви има неочакван резултат…
  - ако знаете, как да се справите с проблема, направете го и продължете. Например:
    - алтернативни данни;
    - по-дълбоко търсене в данните;
    - задна вратичка.
  - ако не, значи тя е малко или много неизползваема и искате да направете всичко спешно и да се евакуирате по най-бързия начин:
    - Оставете изключението да пропагира, но преди това го хванете, за да:
      - добавите информация в log файла, за да може лесно да дебъгнете по-късно;
      - покажете адекватно съобщение за грешка на потребителя;
      - освободите всички резервирани ресурси;
    - Изключението трябва да стигне до най-външния блок на програмата, където да се “разкрие”, или отново да бъде хванато, за да определи генералния exit_code на програмата.

Един бърз пример за логване

```
try:
    me.go_outside()
except MadWifeError:
    me.log('Да се знае - днес не бях пуснат да излизам.')
    raise
```

- raise просто ще хвърли същото изключение, т.е. единствената разлика е, че добавяме log

Добре, а кога да (не) try/except-вам? 4/4

- Ако зависите от външни данни, външен интерфейс, застраховайте се, че получавате правилните данни
  - Нека това не важи за домашните, които са по-скоро Design By Contract - ако дадеш правилни данни, давам правилен резултат, но иначе не гарантирам.
    - Те в общия случай са фрагменти от код, който би бил част от голям продукт, чийто инпут вие самите контролирате.
  - Всяка заявка към отдалечен сървър може да не завърши с това, което очаквате. Подсигурете се, че сте готови за това.
  - Всяко четене/писане на файл може да бъде проблем (липса на права, пълна файлова система, резервиран ресурс…).
  - Всеки свободен вход от потребителя е потенциален проблем.
  - Достъп до външна база данни може да има неочаквани резултати.

Говорейки за работа с файлове...

- Да опитаме да обърнем реда на редовете на файл

```
try:
    source_file = open(src, 'r')
    buffer = []
    try:
        buffer = source_file.readlines()
    finally:
        source_file.close()

    target_file = open(target, 'w')
    try:
        for line in reversed(buffer):
            target_file.write(line)
    finally:
        target_file.close()
except IOError:
    print("Нещо се счупи.")
```

Ех, лошо,
ех, лошо
светът е устроен!

А може, по-иначе може...

ПЕСЕН ЗА ЧОВЕКА от Никола Вапцаров

А може, по-иначе може...

```
try:
    with open(src) as source_file:
        buffer = source_file.readlines()
    with open(target) as target_file:
        for line in reversed(buffer):
            target_file.write(line)
except IOError:
    print("Нещо се счупи.")
```

with гарантира, че файлът ще бъде затворен автоматично.

Learn With with me

```
with израз [as име]:
    блок
```

- Резултатът от израза се нарича Context Manager
- Изпълнява се метода `__enter__()` на CM и резултатът се записва в името след **as**
- Изпълнява се блока
- Ако е настъпило излючение, се изпълнява `__exit__(type, value, traceback)` на CM
- Ако не е настъпило излючение, се изпълнява `__exit__(None, None, None)` на CM

With with With

```
with foo() as f, bar() as b:
    ...
```

е същото като

```
with foo() as f:
    with bar() as b:
        ...
```

Нагледно

```
with open('/etc/passwd') as source_file:
    buffer = source_file.readlines()
print('Done!')
```

е същото като

```
source_file = open('/etc/passwd').__enter__()
try:
  buffer = source_file.readlines()
  source_file.__exit__(None, None, None)
except Exception:
  source_file.__exit__(*sys.exc_info())
print('Done!')
```

Един пример за СМ клас...базиран на "Ало Ало"

>>Ало-ало, тук Нощен ястреб.

>>Лондон, предавам закодирано съобщение:
>>Английските летци са на лекция във ФМИ.
>>Край.

<<Тук Лондон. Прието. Край.

>>Лондон, предавам закодирано съобщение:
>>Никой от тях не е гледал сериала и не разбират за какво говоря.
>>Край.

>>И точка.

- Всеки разговор започва с "Ало-ало, тук Нощен Ястреб";
- Всяко съобщение започва с "Лондон, предавам закодирано съобщение";
- Всяко съобщение завършва с "Край";
- Всеки разговор завършва с "И точка".

Да дефинираме слушателя (Лондон) набързо

```
class Recipient:

    def __init__(self, name):
        self.name = name

    def await_response(self):
        return f"Тук {self.name}. Прието. Край."
```

- Слушателят е клас, който притежава име (в нашия случай това ще е Лондон).
- И също така може да бъде помолен за отговор, за което е дефинирана функцията `await_response`

Да дефинираме мениджър за провеждане на разговор

```
class AlloAlloConversation:

    def __init__(self, name):
        self._name = name

    def __enter__(self):
        print(f"Ало-ало, тук {self._name}.")
        return Recipient("Лондон")

    def __exit__(self, type, value, traceback):
        print("И точка.")
```

- Мениджърът е клас, който получава името на говорителя (в нашия случай - Рене /Нощен Ястреб/).
- Мениджърът е инструктиран да се представи при започване на разговор и да върне слушател.
- Мениджърът е инструктиран да завърши разговора с “И точка”

Да дефинираме мениджър за изпращане на съобщение

```
class AlloAlloMessage:

    def __init__(self, recipient_name):
        self._recipient_name = recipient_name

    def __enter__(self):
        print(f"{self._recipient_name}, "
              "предавам закодирано съобщение:")

    def __exit__(self, type, value, traceback):
        print("Край.")
```

- Мениджърът е клас, който получава името на слушателя (в нашия случай - Лондон).
- Мениджърът е инструктиран да се обърне към слушателя преди предаване на съобщение
- Мениджърът е инструктиран да завърши съобщението с “Край”

Да сглобим всичко

```
with AlloAlloConversation("Нощен ястреб") as recipient:
    with AlloAlloMessage(recipient.name):
        print('Английските летци са на лекция във ФМИ.')
    print(recipient.await_response())
    with AlloAlloMessage(recipient.name):
        print('Никой от тях не е гледал сериала и не разбират за какво говоря.')
```

```
>>Ало-ало, тук Нощен ястреб.
  >>Лондон, предавам закодирано съобщение:
    >>Английските летци са на лекция във ФМИ.
  >>Край.
  <<Тук Лондон. Прието. Край.
  >>Лондон, предавам закодирано съобщение:
     >>Никой от тях не е гледал сериала и не разбират за какво говоря.
  >>Край.
>>И точка.
```

- Пестим еднообразни операции преди всеки разговор и всяко съобщение, делегирайки ги на мениджърите
- Уверяваме се, че в случай на неочакван проблем (Хер Флик нахълта в стаята), ще завършим съобщението и разговора със съответните кодови думи.

Край по темата. Малко организационни въпроси...

- Всички лекции вече са налични и на сайта - подменюто “Материали”.
- Ще получите анкета, с която се надяваме да получим обратна връзка за впечатленията ви от курса.
- Ще получите ново домашно, което е базирано на ООП - очакваме да отнеме по-малко време от предишното, но поради следващата точка, ще дадем една идея повече време за него.
- Първото контролно ще е на 15 ноември (след 12 дни).
- Противно на това, което пише на сайта - няма как да се свържете с нас през fmi@py-bg.net.
- Ползвайте личните ни имейли (можете да ги видите през user-ите в сайта).
- Не ни пращайте dick pics, моля.

Въпроси?