08. Изключения и with

25 декември 2024

DEC 25 == OCT 31

Ние ви питахме, вие отговорихте

- Първо - благодарим!
- Къде за позитивните думи, къде за конструктивната критика - оценяваме и двете!
- А сега да си поговорим малко за резултатите от анкетата
- *(бел. ред. - няма да обсъждаме **всеки** един от въпросите)*

“Скоростта, с която тече курсът”

- За 25% от вас курсът се движи прекалено бързо
- Процентът не е притеснителен, но все пак ни е полезно да го знаем
- Какво ще направим по темата - по-натам

- *П.П.*
- *Има и един, на когото му е прекалено бавно…*
- *Който е от миналата година…*
- *И знаем кой си, господин CamelCase!*

“Знанията, нужни за решаване на домашните”

- Около 30% от вас имат усещането, че домашните изискват набор от знания / похвати, който е по-голям от преподаваното на лекции
- Първо - въпрос...

“Знанията, нужни за решаване на домашните”

- Около 30% от вас имат усещането, че домашните изискват набор от знания / похвати, който е по-голям от преподаваното на лекции
- И това може би е така
- И ние сме окей с това
- Домашните се стремят да тренират фундаменталният набор от знания придобити през изминалите лекции и смятаме, че той е достатъчен...
- Но също така да ви накарат да мислите и четете
- А обратната връзка е нашият опит да ви споделим по-добрите алтернативи, било то от гледна точка на езика или концептуално, не да ви “правим забележки”
- Не смятаме, че да ви държим за ръцете е най-добрата стратегия в дългосрочен план
- Също така няма как да ви обясним всеки метод на всеки обект в стандартната библиотека
- Не си измиваме ръцете с горното, просто ви споделяме нашата мотивация
- Затова едни от първите неща, които ви споделихме бяха:
  - `dir()`
  - `help()`
  - Python 3.13.0 documentation

И все пак

- Ако всичко дотук не ви убеди, че това не е проблем
- Разчитаме на вас да ни питате:
  - В почивките
  - По време на лекции
  - След лекции
  - Във форума
  - В коментарите
  - По телефона
  - Късно вечер, когато сме легнали да четем книга и се чуе тропане по вратата, и “О”, някой студент е дошъл да си говорим за дъндърите, които предефинират достъпа до атрибути
  - Да, толкова сме committed
- Няма да ви кажем как да си напишете домашното от-до, но...
- Ако имате казус, ако се чудите как работи нещо, кой е най-добрият подход в конкретна ситуация, дали има по-добра алтернатива, на това, което сте написали или просто хинт за справяне с някакъв проблем - насреща сме
- Много от вас го правят, следвайте техния пример
- Ако искате да си говорим за музика - също
- И за всичко, really
- Миналата година колегите ви не ни “оставяха на мира”
- Тази година ни е една идея по-самотно ;(

“Настроението, с което се провеждат лекциите”

- 2-ма души отговориха с “Не ми се слушат тъпите ви шеги”
- ;(

“Тема, която да дообясним”

- Декоратори и тяхното по-мащабно приложение
- Декораторите, които бяха най-сложният материал досега
- Обхвата на видимост / живота на променливите
  (бел. ред. - познайте покрай коя лекция беше това)
- Тия декоратори...
- Ако не дообясните декораторите няма никога повече да ви проговоря и ако ви видя по улицата ще ви подминавам без да ви погледна дори, гниди такива
- *(освен ако не ги обясните отново, тогава сме окей, пичаги сте)*

Добре де, това последното може би го нямаше в обратната връзка, но нямаше да се учудим безкрайно.

Close second - OOP

- Което е очаквано, предвид това колко е мащабна темата, че е сравнително прясна и че още не сте имали шанс да го упражните с домашно *(вече имате)*

Ениуей, вие отговорихте, ние чухме!

- Следващите две лекции няма да взимаме нов материал *(такъв, който да го има на контролното)*
- Въпреки това, ще бъдат arguably по-важни от предишните
- 05.11 - OOP workshop:
  - Ще пишем код from scratch, по време на лекция, заедно
  - Ще видим с какви проблеми ще се сблъскаме и как да ги решим
  - Все пак ще сме ограничени от 2-часовият прозорец с който разполагаме, така че не очаквайте нещо грандиозно като краен резултат
  - *Освен това 40-70 души пишат код значително по-бавно от един-двама*
- 07.11 - “Общи” приказки / преговор
  - Ще се върнем на темите, на които може би не сме обърнали достатъчно внимание
  - Да, декоратори
  - Но не само
  - За целта ще ви дадем възможност да ни зададете допълнителни въпроси преди лекцията тогава *(не искаме да ви спамим с анкети, но това ще е най-практичният вариант, ако нямате въпроси - просто игнорирайте)*
  - И както винаги - ако нещо изникне по време на лекция ще си говорим и за това
- Това би следвало да помогне и с проблема “курсът се движи прекалено бързо”

“Една тема, която да покрием в следващите лекции”

- AI семейството (ML, невронни мрежи, data science)
- math и numpy
- Web
- Структуриране на проект
- OOP и design patterns
- Метапрограмиране *(Коце, ти ли си?!)*

За целта сме ви приготвили

- {
  - Метаобектен протокол и Метапрограмиране *(by който дои)*
  - Django 1 & 2 *(by Georgi)*
  - NumPy & Pandas *(by Ani)*
  - NLP and Building your own embeddings *(also by Ani, because Ani sciences better than us)*
  - Лекция с описателното име “Дизайн” *(by Viktor)*
- }

Дисклеймър 1: Редът не е гарантиран.

Дисклеймър 2: Темата “Структуриране на проект” ще трябва да набутаме някъде измежду останалите неща *(реално беше насочено към web, а това сме покрили при Django лекциите)*.

Дисклеймър 3: Тази година няма raging desire за game development лекция, но гледайки как се движим - така или иначе може и да не остане време.

По отношение на шегите

- Няколко души не са доволни от шегата с вените на сникърса
- И ние не бяхме сигурни дали не е прекалено
- Тествахме границите, стана ни ясно
- Понякога сме твърди в позициите си, но не сме инатливи
- С други думи - note taken, ще гледаме да сме по-меки от вече дефинираната граница

Disclaimer

- Имайте предвид, че в лекцията след тази е много вероятно да има шега, включваща думата “пенис”
- Държим да отбележим, че “пенис” е медицински термин:

- Не го правим само за да казваме “пенис”, а е важна метафора, свързана с един от похватите, които ще обясним
- Не ни cancel-вайте

По отношение на задачите

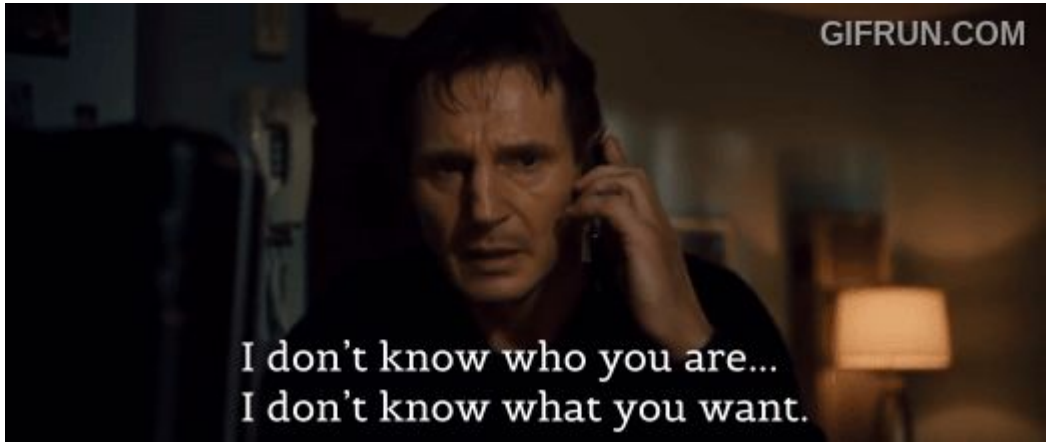
- Немалко от вас искат повече задачи
- Не повече домашни и предизвикателства, които да оценяваме, а такива, които да си решават самостоятелно, с цел практика и задълбочаване на знанията
- Много ми хареса следната идея:
  - “Знам, че изисква много време, но ще е полезно след всяка лекция да се качват като малки задачи за упражнение на показания материал. Не е нужно да са за точки и да ги проверявате, а просто след няколко дни да качвате файл с примерни решения. Мисля, че така теорията от лекции по-лесно ще бъде възприемана и упражнявана.”
- Нямате проблем!
- Идните дни ще получите *(вероятно във форума)* **много** дребни задачи - няма да са безкрайно задълбочени, няма да са безкрайно добре обмислени, няма да сме писали тестове за тях
- На който му се решава - ще има възможност да си решава, а във форума ще имате възможност да задавате въпроси или директно да споделяте решения в търсене на обратна връзка
- След няколко дни ние ще споделим наши решения на тези задачи с малко бележки под линия

По отношение на сникърсите

- Искате:
  - Повече
  - По-здравословни *(с по-високо съдържание на протеин)*
- За първото - предната лекция раздадохме бая, но не можем да ви изхраним всичките!
- За второто - дадено!
- *(другия път)*

И за финал - малко неподправени цитати

- *“Да”* (на въпроса - “Има ли нещо, което ти се иска да правим по различен начин”)
- *“[...] не се усеща като типичен фми курс (разбирайте не поражда суицидни помисли (засега :)))”*
- *“През изминалите теми е имало 2-3 момента на java hate. По правилно би било ако тези случки зачестеят.”*
- *“Очакваме жигосаните кренвирши 😛”*
- *“Искам 1vs1 с Виктор”*



Не знаех как да кръстя този слайд

- Искаме да ви научим на възможно най-много Python
- И да си изкарате добре
- И ние да си изкараме добре
- Невинаги трите неща се случват едновременно, и за всички
- Няма как, и ние се учим
- Може да не сме коментирали част от по-индивидуалните парчета обратна връзка, но това не значи, че не сме ги отразили
- Ако все пак ви притеснява нещо, което вие сте отбелязали, а ние не сме адресирали - насреща сме да го изговорим

Какво е значението на...

- `__getitem__`
- `__setitem__`
- `__getattr__`
- `__setattr__`
- `__getattribute__`

Кое е правилното?

```python
# Вариант 1

class Ass:
    pass


class Teacher:

    def __init__(self):
        self._ass = Ass()
```

```python
# Вариант 2

class Ass:
    pass


class Teacher(Ass):
    pass
```

Какво се случва тук?

```
class Речник(dict):

    име = 'речник'

    def __getitem__(self, name):
        return "Не знам, брат. Ти си знаеш!"

речник = Речник()
речник['име'] = 'стойност'
print(речник.име) # ?
print(речник['име']) # ?

'речник'
'Не знам, брат. Ти си знаеш!'
```

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

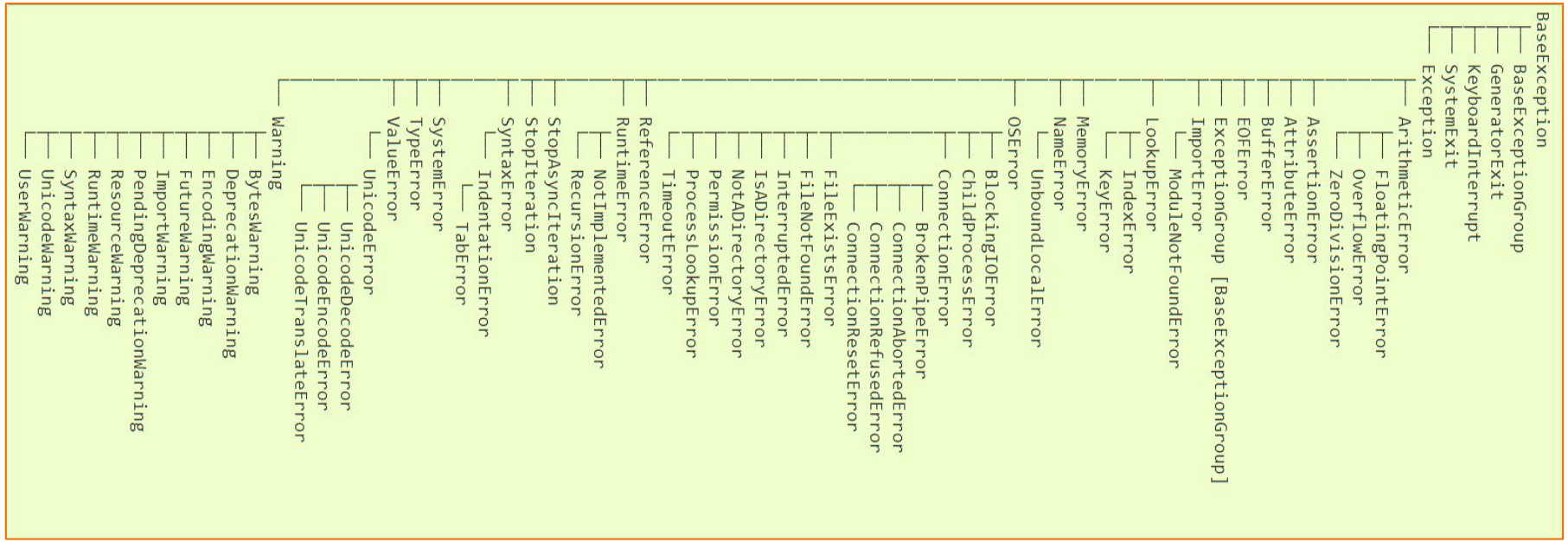
Но какво всъщност представлява try/except?

- 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(без 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

'Аз'.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 с 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 премените 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.
```

BaseException vs Exception

- Всички изключения в Python наследяват BaseException.
- Но вие не искате да наследявате вашите от BaseException, а искате да наследявате от Exception

```
- BaseException
  |- KeyboardInterrupt
  |- SystemExit
  |- Exception
     |- (all other current built-in exceptions)
```

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/except` 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 логваме

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.

Ало-Ало, влиза Нощен Ястреб!

Заслужава си!

Продавачи на лук

Продавачи на лук

Продавачи на лук

Продавачи на лук

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 CM [as име]:
   блок
```

- `CM` (context manager) - инстанция на клас, който имплементира протокола за контекстен мениджър
  - Има имплементиран `__enter__`
  - Има имплементиран `__exit__`
- Изпълнява се метода `__enter__()` на *CM* и резултатът се записва в името след `as`
- Изпълнява се блока
- Ако е настъпило излючение, се изпълнява `__exit__(exc_type, exc_val, exc_tb)` 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 прост пример от github профила на Русалов

```
# execute = Някаква магия, която използва subprocess
class Hack:

    def __init__(self, pswd):
        self._pswd = pswd

    def __enter__(self):
        execute('sudo su', stdin=self._pswd)

    def __exit__(self, *_):
        execute('rm -rf *.log && exit')

hack = Hack('secret_password')

with hack:
    execute('bash malicious_script')
```

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, exc_type, exc_val, exc_tb):
        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, exc_type, exc_val, exc_tb):
        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.
```

Извод

- 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.

Въпроси?