“Общи” приказки

6 април 2026

Първо - контролното!

- Йоан ви каза достатъчно за резултатите, аз искам да ви споделя кои се оказват “трудните” въпроси

Декориране

- **Въпросът:**

- Процент верни отговори: **31.6%**
- Тук имахте и неумишлен жокер, двата стринга са различни - `"has been"` vs `"was"`
- Истината е, че ще хвърли грешка
- Защо?

28.
```
def log_execution(func):
    def decorated():
        result = func()
        print(f'Function "{func.__name__}" has been executed!')
        return result
    return decorated

@log_execution
def square(num):
    return num ** 2

square(25)
```

Какво ще изведе горния код?

А: Ще хвърли грешка
Б: 'Function "square" was executed!'
В: 'Function "decorated" was executed!'
Г: 'Function "log_execution.<locals>.decorated()" was executed!'

Истинност

- **Въпросът:**

- Процент верни отговори: **26.3%**
- Най-вероятната грешка е да сте отбелязали `set([])`
- Празните колекции са false-y, горното е празна колекция
- Всичко празно, или еквивалентно на 0 е false-y
- Единственото, което остава е `(i for i in range(0))`
- Привидно също е празно, но това е само чак след като се оцени... А то е lazy

27. Кое от следните неща **е** истина?

А: None
Б: ""
В: 0j
Г: set([])
Д: (i for i in range(0))
Е: bool([i for i in dict()])
Ж: {}

Дълбочина

- **Въпросът:**

- Брой верни отговори: **1**
- Уловката тук отново са референциите
- `matrix[:]` прави “плитко” копие на колекцията
- Нека онагледим
- За любознателните - `copy.deepcopy`

10. matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
matrix_copy = matrix[:]
matrix_copy[2][1] = -10
matrix_copy[1] = [-4, -5, -6]
Каква е стойността на matrix?

А: [[1, 2, 3], [-4, -5, -6], [7, 8, 9]]
Б: [[1, 2, 3], [4, 5, 6], [7, -10, 9]]
В: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
Г: [[1, 2, 3], [-4, -5, -6], [7, -10, 9]]

Грешка

- **Въпросът:**

26. Каква е разликата между sorted(some_list) и some_list.sort()?

А: sorted е по-бърз от sort
Б: sort ползва линейно количество памет
В: sorted връща нов, сортиран списък, а sort сортира списъка in-place
Г: Няма разлика

- Брой верни отговори:
  1
- Странно, а?
- Е, оказа се, че ние сме го маркирали грешно в системата 😅
- Опа...
- Оправили сме грешката и сме ви освежили точките, но предвид това, че точките се нормализират - не повече от 5-6 човека следва да усетят разликата

Второ - анкетата!

- 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 **8.7%** от вас курсът се движи прекалено бързо, за останалите **91.3%** - с точното темпо
- Процентът е супер!
- С оглед на малкия процент, за момента няма да правим екстра лекции с преговори и прочие, така или иначе сме предвидили някои от следващите сбирки да са live coding, което ще помогне да се затвърди материала
- Все пак, за тези 8.7% - “досаждайте” ни повече в почивките, говорете с колегите си, пишете повече код

“Знанията, нужни за решаване на домашните”

- Около **25%** от хората имат усещането, че домашните изискват повече знания от преподаденото на лекции
- И това е напълно окей
- Понякога ви даваме инструментите и им обръщаме внимание, понякога ви даваме индикация за съществуването им
- Не навлизаме във всяко едно нещо в детайли, но би следвало когато ви се напаснат достатъчно парченца от пъзела, да можете на базата на дедукция да имате очакване как работи езикът и къде какво можете да търсите
- Това с времето, разбира се
- Идеята е - рядко ще ви дадем домашно, което изисква похвати, които не сме обяснили концептуално. Да, може да се случи да не сме ви показали **точно този метод**, но сме ви дали достатъчно контекст да предположите, че съществува
- А когато е наистина нещо ново - казваме ви в прав текст какво да ползвате
- И финално - обратната връзка е нашият опит да ви споделим по-добрите алтернативи, било то от гледна точка на езика или концептуално, и когато ви предложим нещо, което досега не сме преподавали - това не значи, че решението ви е грешно... Просто ви даваме още инструменти 🙂

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
- Ако искате да си говорим за изкуство, филология, Bad Dragon колекциите - също
- И за всичко, really

“Настроението, с което се провеждат лекциите”

- **21.7%** от вас искат още тъпи шеги
- Ще се постараем!
- Например, знаете ли какво е “reverse exorcism”?

“Една тема, която да покрием в следващите лекции”

- Основно тези двете изскочиха:
  - Web
  - NumPy / Pandas / ML
- Имаше и още нещо, което не искаме да интерпретираме грешно, затова подканваме човекът, който го е написал да дойде в някоя почивка да си поговорим по темата:
  - Нещо по девопско например как да свържа някакъв мой проект към съществуващо апи

“Тема, която да дообясним”

- *“Може би ще е добре пак да се върнем и преговорим декораторите”*
- *“Dekoratori no poveche primeri”*
- *“Декоратори”*
- *“Може би декораторите”*
- *“Лично за мен декораторите са ми от по-трудния материал за усвояване”*
- *“Декораторите биха били не лоша тема за преговор”*
- *“Декоратори може малко”*
- *“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.

Ениуей, вие отговорихте, ние чухме!

- След малко ще си поговорим още малко за декораторите, с примери
- Освен това, искаме да адресираме едно конкретно парче обратна връзка:
  - *“[...] на някои по-сложни [теми] се отделя по малко от нужното да се разберат. Като че ли не е разпределено правилно времето на лекциите и доста пъти не достига да се изглади всичко”*
- По отношение на втората част - абсолютно
- Всяка година местим, променяме, добавяме материал
- Но по-важното - всяка година получаваме различни въпроси и различно усещане за това кое от казаното land-ва, и кое - не
- Но нашата преценка не винаги ще се окаже правилна
- Затова разчитаме на вас да ни:
  - Задавате въпроси
  - Кажете на края на лекцията, когато ви попитаме дали имате въпроси - “Прекалено много, може ли следващия път да дообясните X?”
  - Кажете в началото на следващата лекция - “Сякаш не отделихме достатъчно време на X, може ли да се върнем малко на темата?”
- Много харесвам една история на Саймън Синек по темата

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

Та, да си дойдем на очевидно любимата тема

Декораторите като design pattern

- Позволяват ни да надградим функционалността на един обект, “опаковайки” го по някакъв начин
- В контекста на питонските декоратори - надграждаме функционалността на функция с допълнително поведение
- Това, вярвам, ви е ясно
- Объркващото идва когато започнем да правим `@fun1("arg1")("arg2")("arg3")`
- Едно по едно

Преводи

- Да речем, че сме понаписали определен брой функции, които връщат като резултат низ
- Обаче ни идва нагости братовчед ни от Щатите
- А низовете ни са на Български!
- Нека да го улесним
- *П.П. Това далеч не е най-добрият вариант за имплементиране на internationalization / localization*

Нашата функция

Нека да дефинираме проста функция, която връща низ:

```
def кон():
    return "отиде коня у ряката"

кон() # 'отиде коня у ряката'
```

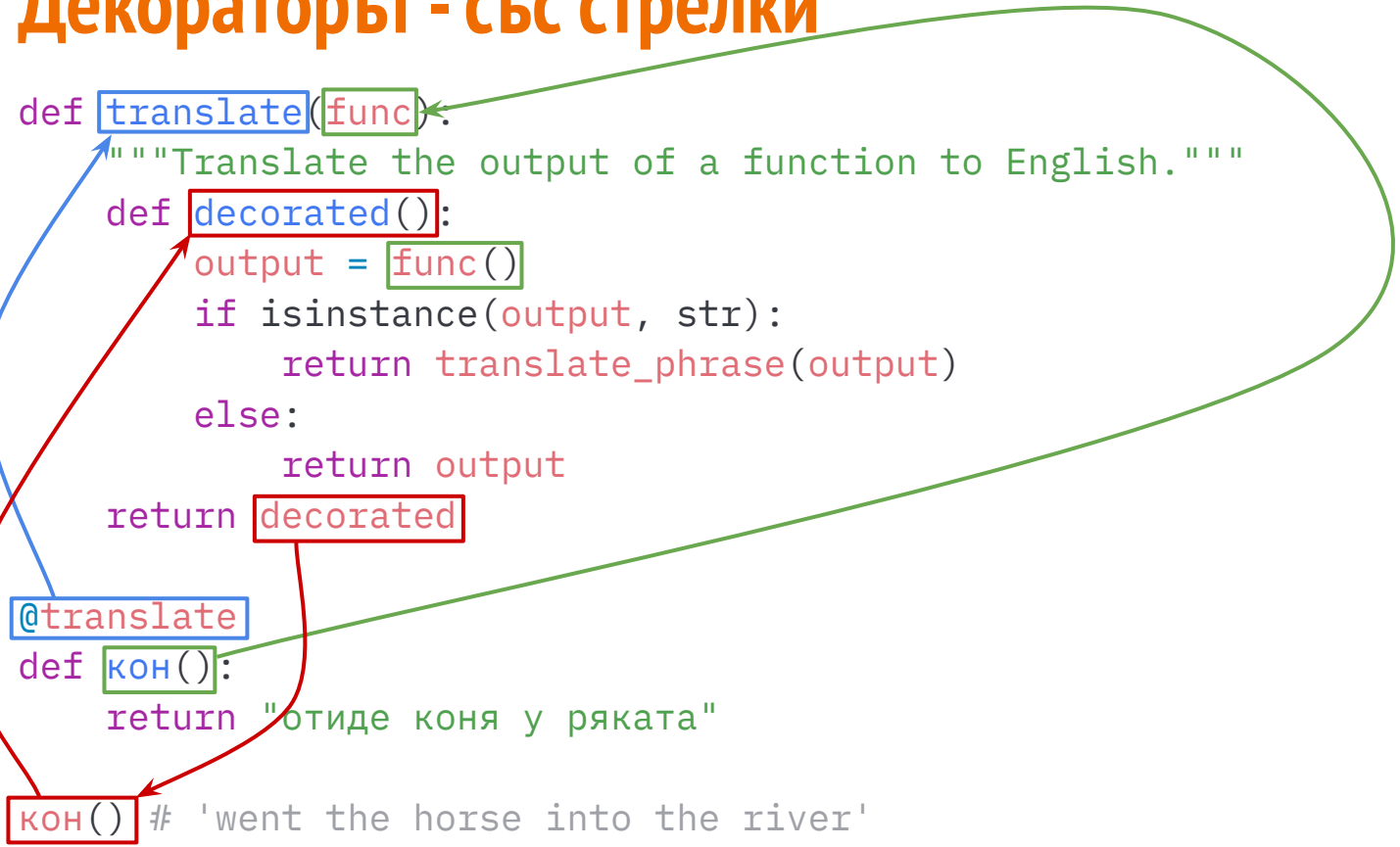
Искаме следното:

```
@translate
def кон():
    return "отиде коня у ряката"

кон() # 'went the horse into the river'
```

Декораторът

```
def translate(func):
    """Translate the output of a function to English."""
    def decorated():
        output = func()
        if isinstance(output, str):
            return translate_phrase(output)
        else:
            return output
    return decorated

@translate
def кон():
    return "отиде коня у ряката"

кон() # 'went the horse into the river'
```

Декораторът - със стрелки



```
def translate(func):
    """Translate the output of a function to English."""
    def decorated():
        output = func()
        if isinstance(output, str):
            return translate_phrase(output)
        else:
            return output
    return decorated

@translate
def кон():
    return "отиде коня у ряката"

кон() # 'went the horse into the river'
```

translate_phrase

За пълнота, ето ви спомагателната функция:

```
SENTENCE_TRANSLATIONS = {"отиде коня у ряката": "went the horse into the river"}
WORD_TRANSLATIONS = {"отиде": "went", "коня": "the horse",
                     "у": "into", "ряката": "the river"}

def translate_phrase(phrase):
    full_translation = SENTENCE_TRANSLATIONS.get(phrase)
    if full_translation is None:
        words = phrase.split()
        words = [WORD_TRANSLATIONS.get(word, word) for word in words]
        full_translation = " ".join(words)
    return full_translation
```

Забележка - спрямо промените, които ще правим, тази функция също ще се променя. Приемайте я като абстракция, за целите ни не е важна нейната имплементация.

По-универсално решение

Добре, а ако искаме да декорираме функции, които приемат входни параметри?

```
@translate
def кон(number_of_horses):
    return f"отидоха {number_of_horses} коня у ряката"

кон(5) # ???
```

```
Traceback (most recent call last):
  File "D:/Dev/FMI/Python/decorators_xample_args_unfixed.py", line 27, in
<module>
    print(кон(5))
TypeError: translate.<locals>.decorated() takes 0 positional arguments but 1
was given
```

Защо?

```
def translate(func):
    """Translate the output of a function to English."""
    def decorated():
        output = func()
        if isinstance(output, str):
            return translate_phrase(output)
        else:
            return output
    return decorated

@translate
def кон(number_of_horses):
    return f"отидоха {number_of_horses} коня у ряката"

кон(5) # ....decorated() takes 0 positional arguments but 1 was given
```

Решението

```
def translate(func):
    """Translate the output of a function to English."""
    def decorated(*args, **kwargs):
        output = func(*args, **kwargs)
        if isinstance(output, str):
            return translate_phrase(output)
        else:
            return output
    return decorated

@translate
def кон(number_of_horses):
    return f"отидоха {number_of_horses} коня у ряката"

кон(5) # 'went 5 the horse into the river'
```

Братовчедът от Япония

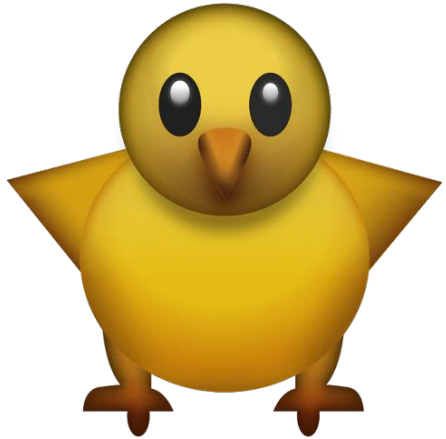
Добре, а ако имаме функции, изходът от които искаме да превеждаме на различен език?

```
@translate("jp")
def кон(number_of_horses):
    return f"отидоха {number_of_horses} коня у ряката"

кон(5) # ???
```

```
Traceback (most recent call last):
  ...
  File "D:/Dev/FMI/Python/decorators_xample_jp_not_callable.py", line 16, in
decorated
    output = func(*args, **kwargs)
TypeError: 'str' object is not callable
```

Защо?

```
def translate(func):
    """Translate the output of a function to English."""
    def decorated():
        output = func()
        if isinstance(output, str):
            return translate_phrase(output)
        else:
            return output
    return decorated

@translate("jp")
def кон(number_of_horses):
    return f"отидоха {number_of_horses} коня у ряката"
```

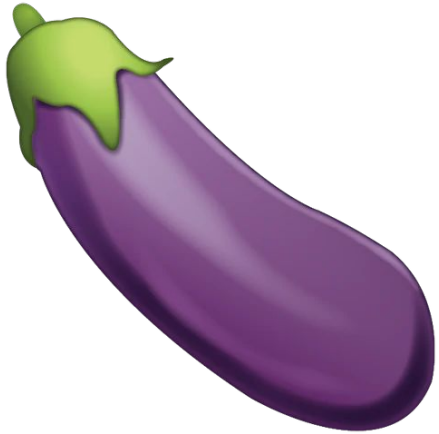


Решението

```
def translate(language="en"):
    """Translate the output of a function to a specific language."""
    def translate_to(func):
        def decorated(*args, **kwargs):
            output = func(*args, **kwargs)
            if isinstance(output, str):
                return translate_phrase(output, language)
            else:
                return output
        return decorated
    return translate_to

@translate("jp")
def кон(number_of_horses):
    return f"отидоха {number_of_horses} коня у ряката"


кон(5) # '行った 5 馬 で 川'
```

И за финал

```
@translate("вр")
def народна_мъдрост():
    return "Не се опитвай да угодиш на всички."

народна_мъдрост()
# 'Не моа са аресаш на сички. Не си рикия!'
```



Един бонус - истинският деКОРатор

```
def de_кор_ator(fun):
    def clean_fun():
        return fun().replace('кор', '***')
    return clean_fun

@de_кор_ator
def student_statement():
    return "Да търпиш глупостите на лекторите е голям кор!"

print(student_statement()) # 'Да търпиш глупостите на лекторите е голям ***!'
```

Знаем, от декораторите боли глава

12. Част 2 - Среди

6 април 2026

Средата е важна

Програмата ви има нужда от правилната среда за да работи:

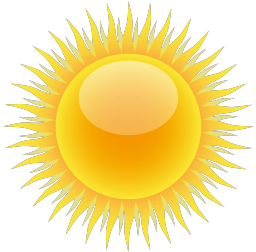
- Операционна система
- Файлова система
- Интерпретатор
- Библиотеки
- Връзка с интернет
- ...

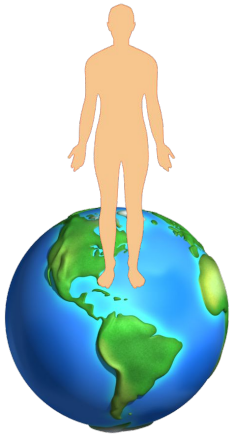
По-лесно е да контролираме стабилна средата, отколкото да преправим програмата така, че да работи във всяка среда.

Как да подготвя средата?

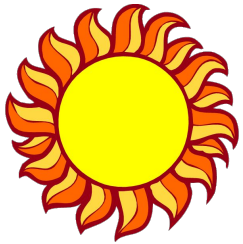
- Желязо
- Виртуална машина
- Докер
- Виртуална среда

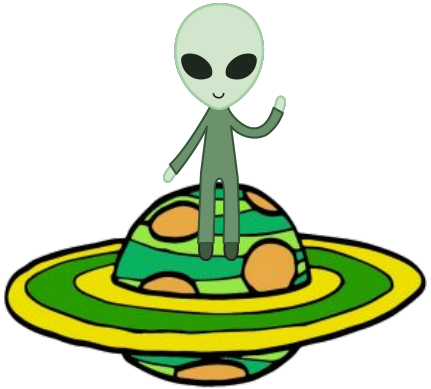
Да поиграем на Господ





- Различните програми имат нужда от различни среди
- Различните форми на живот също





Ако искаме да предоставим нова среда, можем...

- просто да създадем нова Вселена

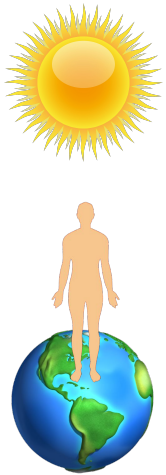
Което е съпоставимо с това да...

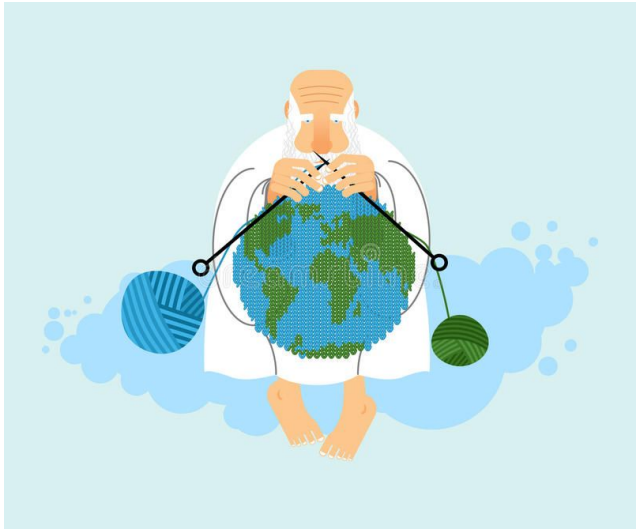
- осигурим ново желязо

- но това е доста скъпо решение

Ако искаме да предоставим нова среда, можем...

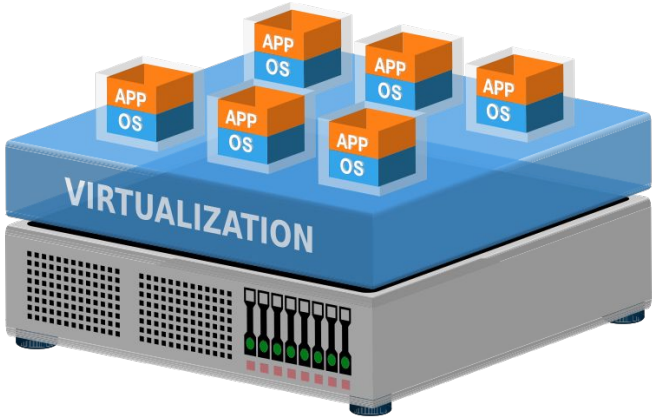
- просто да създадем ново Слънце и нова планета





Което е съпоставимо с това да...

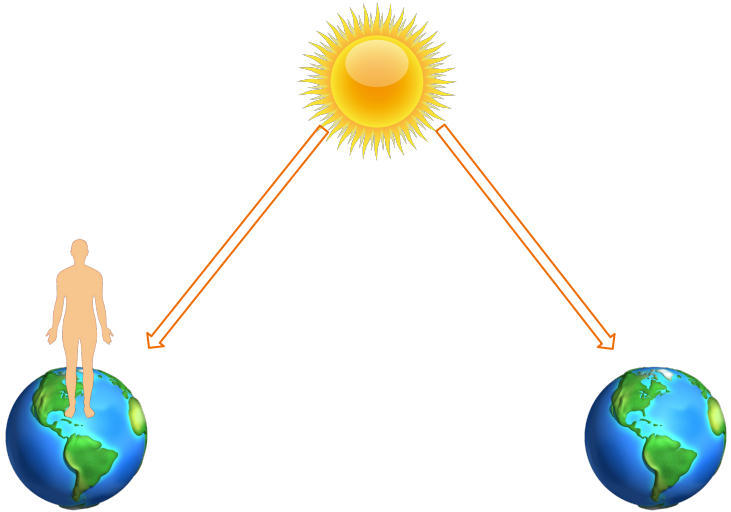
- създадем виртуална машина във вече съществуващо желязо



- няма нужда от нов хардуер, но всяка виртуална машина притежава своя собствена операционна система, което изисква ресурс

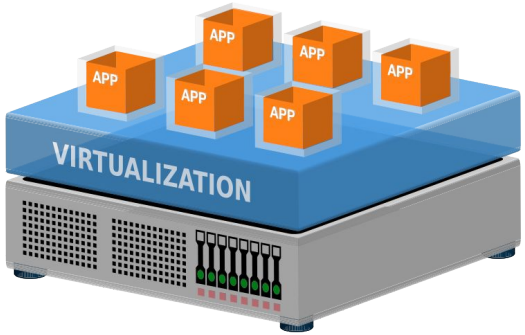
Ако искаме да предоставим нова среда, можем...

- просто да създадем нова планета, използвайки същото слънце



Което е съпоставимо с това да...

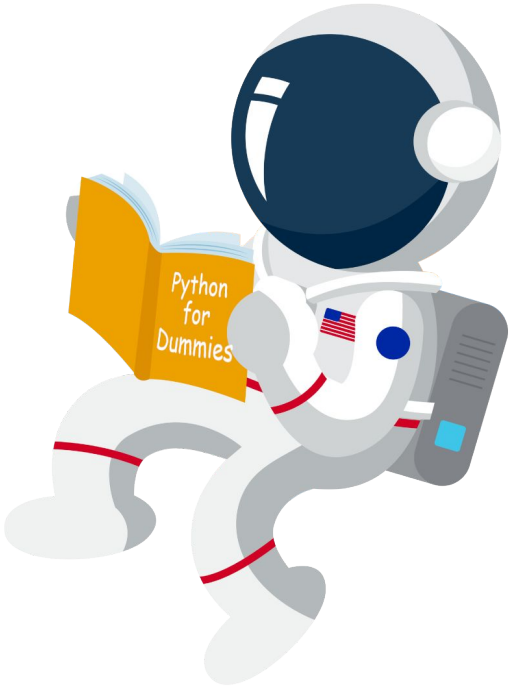
- използваме докер контейнер



- няма нужда от нов хардуер и няма нужда от отделна операционна система за всеки контейнер, но докерът притежава собствена файлова система, процеси, потребители, мрежа...

Ако искаме да предоставим нова среда, можем...

- просто да връчим на човека един скафандър и да го пращаме навсякъде



Което е съпоставимо с това да...

- използваме виртуална среда

- единственото новосъздадено нещо е Python, с прилежащите си библиотеки, и конкретни настройки по променливите на средата

Променливи на средата (environment variables)

- Предоставят се от операционната система
- Използват се, за да определят различни свойства на средата
- Потребителят може да ги промени, за да контролира средата
- Всеки процес (програма) получава копие от средата и може да я модифицира за лично ползване
- Това до известна степен ни позволява да използваме различни среди за различни програми

Как изглеждат тези променливи?

Windows - $ set

```
PATH=C:\ProgramFiles;C:\Users\Stamat\AppData\Local\Programs\Python\Python310\;
...
TEMP=C:\Users\Stamat\AppData\Local\Temp
HOMEDRIVE=C:
HOMEPATH=\Users\Stamat
…
```

Как изглеждат тези променливи?

Linux/MacOS - $ env

```
PATH=/home/gvkunchev/.local/bin:/usr/local/bin:/usr/bin:/bin:/usr/local/games:
/usr/games
PWD=/home/gvkunchev/cool_stuff
HOME=/home/gvkunchev
SHELL=/bin/bash
…
```

PATH

- *PATH* е променливата, която се използва при извикване на команда
- Когато отворите терминал и напишете `python`, *shell*-ът търси в *PATH*, за да намери какво искате да пуснете
- Това значи ли, че мога да имам инсталирани две различни версии на Python и да контролирам коя се извиква чрез модификация на *PATH*?
- Анджак!

Да го направим под Windows

```
C:\Users\Stamat>python
Python 3.10.7 (tags/v3.10.7:6cc6b13, Sep  5 2022, 14:08:36) [MSC v.1933 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.

>>> quit()

C:\Users\Stamat>echo "ECHO I am fake Python" > Desktop\python.bat

C:\Users\Stamat>set PATH=C:\Users\Stamat\Desktop;%PATH%

C:\Users\Stamat>python
I am fake Python
```

Да го направим под Linux

```
gvkunchev@instance-1:~$ python
Python 2.7.18 (default, Jul 14 2021, 08:11:37)
[GCC 10.2.1 20210110] on linux2
Type "help", "copyright", "credits" or "license" for more information.

>>> quit()

gvkunchev@instance-1:~$ mkdir dummy_python

gvkunchev@instance-1:~$ echo "echo 'I am fake python'" > dummy_python/python

gvkunchev@instance-1:~$ chmod +x dummy_python/python

gvkunchev@instance-1:~$ PATH="dummy_python:%PATH%"

gvkunchev@instance-1:~$ python
I am fake python
```

Да не преоткриваме колелото

```
$ python -m pip install virtualenv
```

- *pip* е модул и като всеки друг модул, той може да се изпълни чрез “`python -m <module>`”
- *pip* е мениджъра за пакети на Python и идва заедно с него по подразбиране
- *pip* използва онлайн хранилища, за да намери и инсталира пакета, който ви трябва
- също така се грижи да инсталира и евентуални зависимости на този пакет
- *virtualenv* е пакет, който ви позволява да създавате виртуални среди през *CLI*

```
Collecting virtualenv
  Downloading virtualenv-20.16.7-py3-none-any.whl (8.8 MB)
     ---------------------------- 8.8/8.8 MB 9.8 MB/s eta 0:00:00
Collecting distlib<1,>=0.3.6
  Downloading distlib-0.3.6-py2.py3-none-any.whl (468 kB)
     ---------------------- 468.5/468.5 kB 9.8 MB/s eta 0:00:00
Installing collected packages: distlib, virtualenv
Successfully installed distlib-0.3.6 virtualenv-20.16.7
```

Създаване и активиране на виртуална среда

```
$ virtualenv django_venv
```

- създава директория с всичко, което е нужно за изолиране на *Python* от глобално инсталираната версия

```
$ django_venv\Scripts\activate (Windows)
$ source django_venv/bin/activate (Linux)
```

- активира средата, променяйки променливите на средата така, че да използвате Python версията от създадената директория

```
$ where python (Windows) # …\django_venv\Scripts\python.exe
$ which python (Linux) # …/django_venv/bin/python
```

- Ако искате да излезете от средата:
  ```
  $ deactivate
  ```

Използване на виртуална среда

- Бидейки във виртуална среда, можем да инсталираме различни пакети за различни проекти
  - Например:
    ```
    $ python -m pip install django=2.0.0
    ```
- Можем дори да го правим за различни версии на Python
- По този начин изолираме зависимостите на проекта от глобалната среда
- Можем да дефинираме всички пакети, от които се нуждаем, в един *requirements.txt* файл, след което просто създаваме виртуална среда, и в нея пускаме:
  ```
  $ python -m pip install -r requirements.txt
  ```
  - `"$ python -m pip freeze" ще изплюе този файл наготово, спрямо текущото състояние на средата`
- Ако нещо се объркa, просто трием една директория

venv

- От известно време *Python* идва с модул за създаване на виртуални среди, наречен “*venv*”
- На повечето места той е готов за използване и се използва точно така, както показахме с *virtuelenv*
- На някои места, обаче, например *Ubuntu*, той самият изисква допълнителни пакети
- Хубавото е, че можете да правите виртуални среди, използвайки Python код

```
import venv

venv.create("/tmp/django_venv")
```

Wheel - стандартът за разпространение на пакети

- Както казахме, “`pip install`“ търси пакети в онлайн хранилище
- Хранилището е https://pypi.org/
- То съдържа *wheel* файлове (**.whl*), които съдържат целия сорс код на даден пакет, плюс мета информация за версии, автори, зависимости и т.н.
- Ако искате да създадете пакет, който да разпространите, това е мястото

Защо Wheel?

- Преди *Wheel* имаше *Egg*, който имаше нужда от подобрения
  - *Egg*, защото... *Monty Python* (*spam and eggs*)
  - а и питоните се раждат от яйца
- *Wheel* е подобрената версия на *Egg - Python Packaging* ***reinvented***
- Освен това *wheel* е името на **кутиите за сирене**
  - Отново *Monty Python* и техният скеч за сиренето
- А и това, което *wheel* прави, е "***roll*** *out software*"

Да направим и ние един wheel файл

```python
# cool_stuff.py
from coolprint import coolprint

def cool_function():
    coolprint("I am cool.")
```

- cool_project
  - __init__.py
  - cool_stuff.py
- setup.py

```python
# setup.py
from setuptools import setup, find_packages

setup(
    name='cool_project',
    version='1.0',
    packages=find_packages(),
    python_requires='>=3.7',
    install_requires=['coolprint']
)
```

Подготовката е направена - да генерираме

- `$ python setup.py bdist_wheel`
- Създава разни директории, но това, което нас ни интересува, е: *dist/cool_project-1.0-py3-none-any.whl*
- {distribution}-{version}(-{build tag})?-{python tag}-{abi tag}-{platform tag}.whl
  - * ABI - Application Binary Interface
- Това е просто *.zip* файл, така че ако искате, можете да го отворите с подходяща програма и да го разгледате

Да направим среда и да инсталираме пакета

- `$ virtualenv coolenv`
- `$ cd coolenv`
- <слагаме .whl файла в директорията>
- `$ python -m pip install cool_project-1.0-py3-none-any.whl`
- `...Successfully installed cool-project-1.0 coolprint-0.0.2`
- Пакетите са инсталирани в *site_packages* на виртуалната среда

```
(coolenv) C:\Users\stamat\Desktop\coolenv>python
>>> from cool_project.cool_stuff import cool_function
>>> cool_function()
```

Виртуална среда

- Ще направим едно ново репозитори в GitHub
- Ще го клонираме локално
- Ще създадем виртуална среда
- Ще инсталираме coolprint
- Ще напишем няколко реда код с него
- Ще направим requirements.txt
- Ще напишем README.md
- Ще качим всичко в GitHub

Въпроси?