

BREAKING HAD

KDO?

Vojta (Discord: #Hackrrr)

KDO?

Vojta (Discord: #Hackrrr)

- IT, Programování

KDO?

Vojta (Discord: #Hackrrrr)

- IT, Programování
- Hacking, CTFs (Hraní i tvorba; ECSC, Team Europe candidate, ...)

KDO?

Vojta (Discord: #Hackrrrr)

- IT, Programování
- Hacking, CTFs (Hraní i tvorba; ECSC, Team Europe candidate, ...)
- Python

KDO?

Vojta (Discord: #Hackrrr)

- IT, Programování
- Hacking, CTFs (Hraní i tvorba; ECSC, Team Europe candidate, ...)
- Python
- Další nerelevantní věci...

CO?

CO?

- Python

CO?

- Python
- Obskurní featury a chování

CO?

- Python
- Obskurní featury a chování
- Divné "implementační detaily"

CO?

- Python
- Obskurní featury a chování
- Divné "implementační detaily"
- Pár "homemade" šíleností (pokud bude čas)

CO?

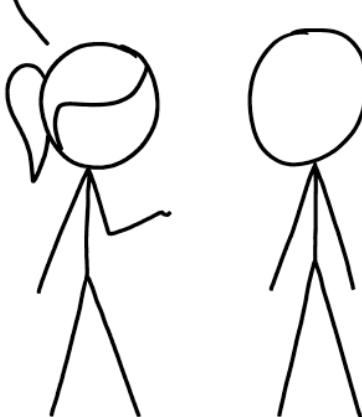
- Python
- Obskurní featury a chování
- Divné "implementační detaily"
- Pár "~~homemade~~" šíleností (pokud bude čas)

CO?

- Python
- Obskurní featury a chování
- Divné "implementační detaily"
- Pár "~~homemade~~" šíleností (pokud bude čas)
- Pravděpodobně mentální újma

SILICATE CHEMISTRY IS SECOND NATURE TO US GEOCHEMISTS, SO IT'S EASY TO FORGET THAT THE AVERAGE PERSON PROBABLY ONLY KNOWS THE FORMULAS FOR OLIVINE AND ONE OR TWO FELDSPARS.

| AND QUARTZ, OF COURSE.
OF COURSE.



EVEN WHEN THEY'RE TRYING TO COMPENSATE FOR IT, EXPERTS IN ANYTHING WILDLY OVERESTIMATE THE AVERAGE PERSON'S FAMILIARITY WITH THEIR FIELD.

Disclaimer: Pokud jste Python nikdy nepsali, tak si tuhle přednášku (asi) tak moc neužijete a pravděpodobně budete relativně často ztraceni.

Disclaimer: Pokud jste Python nikdy nepsali, tak si tuhle přednášku (asi) tak moc neužijete a pravděpodobně budete relativně často ztraceni.

Na druhou stranu (ad XKCD) já dost možná nadceňuji, co normální člověk o Pythonu ví, takže se mě nebojte zastavit...

Disclaimer: Pokud jste Python nikdy nepsali, tak si tuhle přednášku (asi) tak moc neužijete a pravděpodobně budete relativně často ztraceni.

Na druhou stranu (ad XKCD) já dost možná nadceňuji, co normální člověk o Pythonu ví, takže se mě nebojte zastavit...

O existenci `.__globals__` atributu na funkčích přeci ví každý, ale pravda že jak "z venku" měnit lokální proměnné funkce pomocí `.__closure__` asi není tak běžná znalost...

PROČ? (VY)

PROČ? (VY)

- Nevím :)

PROČ? (VY)

- Nevím :)
- Nic co na práci

PROČ? (VY)

- Nevím :)
- Nic co na práci
- Touha zjistit se nepodstané a neužitečné informace o Pythonu

PROČ? (VY)

- Nevím :)
- Nic co na práci
- Touha zjistit se nepodstané a neužitečné informace o Pythonu
- Vědět jak Pythonem dohnat ostatní k šílenstí

PROČ? (JÁ)

PROČ? (JÁ)

- Farmení "Wat?!" reakcí

PROČ? (JÁ)

- Farmení "Wat?!" reakcí
- Možnost terorizovat ostatní mým Pythonem

PROČ? (JÁ)

- Farmení "Wat?!" reakcí
- Možnost terorizovat ostatní mým Pythonem
- PyJails <3

PYJAIL(S)?

CTFkové úlohy typu "Python kód jako vstup, server něco validuje/filtruje a ty se pokus přečíst flag.txt."

```
1 code = input()
2
3 if any(x in code for x in "0123456789"):
4     print("Nope.")
5     exit(1)
6
7 print(eval(code, globals={"__builtins__": {}}))
```

```
()).__class__.__mro__[-True].__subclasses__()([-True-True-True-True])  
. __call__. __globals__["__builtins__"]["__import__"]("os").system("type flag.txt")
```

```
().__class__.__mro__[True].__subclasses__()TrueTrueTrueTrue  
. __call__. __globals__[builtins][import]("os").system("type flag.txt")
```

Čekejte podobně kvalitní kód :)

```
().__class__.__mro__[True].__subclasses__()TrueTrueTrueTrue  
. __call__. __globals__[builtins][import]("os").system("type flag.txt")
```

Čekejte podobně kvalitní kód :)

Řešení PyJailů => zákonitě se začne narážet na oskurnosti.

```
().__class__.__mro__[True].__subclasses__()TrueTrueTrueTrue  
. __call__. __globals__[builtins][import]("os").system("type flag.txt")
```

Čekejte podobně kvalitní kód :)

Řešení PyJailů => zákonitě se začne narážet na oskurnosti.

Přednáška nebudou PyJails, ale na dost věcí jsem narazil právě při řešení PyJailů / hackování Pythonu.

ACTUAL START OF TALK

PYTHON 2

Nejprve trocha historie... Co je špatně na následujícím kódu?

```
1 try:  
2     num = input("Number: ")  
3 except KeyboardInterrupt:  
4     num = -1  
5  
6 print("Your number is " + str(num) + "!")
```

PYTHON 2

Nejprve trocha historie... Co je špatně na následujícím kódu?

```
1 try:  
2     num = input("Number: ")  
3 except KeyboardInterrupt:  
4     num = -1  
5  
6 print("Your number is " + str(num) + "!")
```

`input([prompt])`

Equivalent to `eval(raw_input(prompt))`.

This function does not catch user errors. If the input is not syntactically valid, a `SyntaxError` will be raised. Other exceptions may be raised if there is an error during evaluation.

If the `readline` module was loaded, then `input()` will use it to provide elaborate line editing and history features.

Consider using the `raw_input()` function for general input from users.

```
$ echo 'open("/etc/passwd").read()' | python script.py
Number: Your number is root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
!
```

ZIP

```
$ python3 script.py  
Hello World!
```

ZIP

```
$ python3 script.py  
Hello World!
```

```
$ cat script.py  
#!/usr/bin/python3  
#  
# This is a simple script to demonstrate ZIP.  
# It takes two arguments:  
# 1. A file to compress (script.py)  
# 2. The output ZIP file (output.zip)  
  
# Import the required module  
import zipfile  
  
# Open the ZIP file in write mode  
with zipfile.ZipFile('output.zip', 'w') as zipf:  
    # Add the file to the ZIP archive  
    zipf.write('script.py')
```

ZIP

```
$ python3 script.py
Hello World!

$ cat script.py
X~Z
__main__.pyux
    g;      gUT    g+(+PH?/??/IQT?/??P?/???X~Z?/??
?!__main__.pyux
                                UT   gPKQr
$ file script.py
script.py: Zip archive data, at least v2.0 to extract, compression method=deflate
```

`<script>`

Execute the Python code contained in `script`, which must be a filesystem path (absolute or relative) referring to either a Python file, a directory containing a `__main__.py` file, or a zipfile containing a `__main__.py` file.

If this option is given, the first element of `sys.argv` will be the script name as given on the command line.

If the script name refers directly to a Python file, the directory containing that file is added to the start of `sys.path`, and the file is executed as the `__main__` module.

If the script name refers to a directory or zipfile, the script name is added to the start of `sys.path` and the `__main__.py` file in that location is executed as the `__main__` module.

<https://docs.python.org/3/using/cmdline.html#cmdarg-script>

ZAJÍMAVÉ MODULY

ZAJÍMAVÉ MODULY

- typing, dataclasses, ...

ZAJÍMAVÉ MODULY

- ~~typing, dataclasses, ...~~

ZAJÍMAVÉ MODULY

- ~~typing, dataclasses, ...~~
- `__main__, __future__`

ZAJÍMAVÉ MODULY

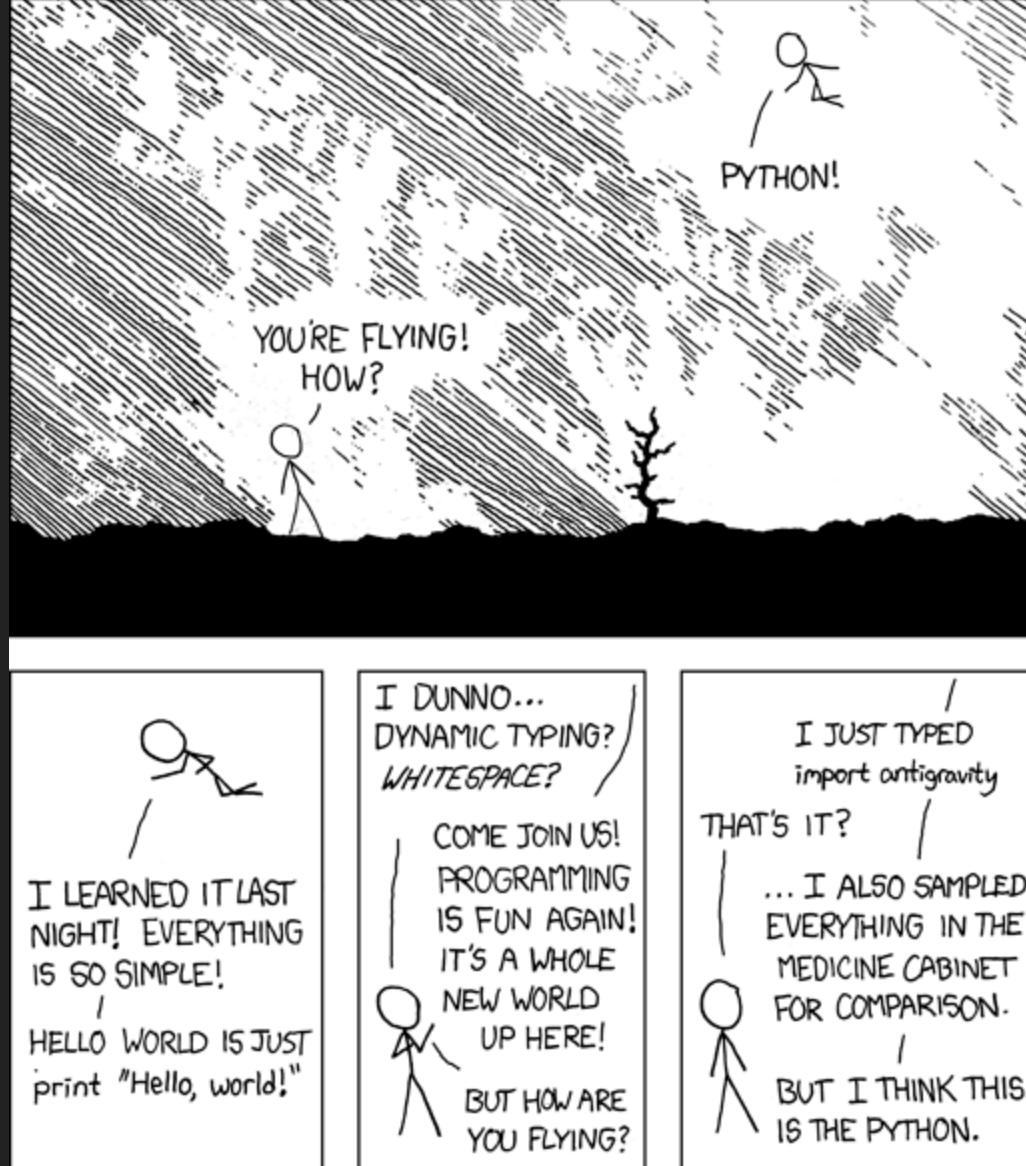
- ~~typing, dataclasses, ...~~
- ~~__main__, __future__~~

ZAJÍMAVÉ MODULY

- ~~typing, dataclasses, ...~~
- ~~__main__, __future__~~
- ~~this~~

```
1 s = """Gur Mra bs Clguba, ol Gvz Crgref
2
3 Ornhgvshy vf orggre guna htyl.
4 Rkcyvpvg vf orggre guna vzcyvpvg.
5 Fvzcyr vf orggre guna pbzcyrk.
6 Pbzcyrk vf orggre guna pbzcryvpngrq.
...
18 Nygubhtu arire vf bsgra orggre guna *evtug* abj.
19 Vs gur vzcyrzragngvba vf uneq gb rkcytva, vg'f n onq vqrn.
20 Vs gur vzcyrzragngvba vf rnfl gb rkcytva, vg znl or n tbbq vqrn.
21 Anzrfcnprf ner bar ubaxvat terng vqrn -- yrg'f qb zber bs gubfr!"""
22
23 d = {}
24 for c in (65, 97):
25     for i in range(26):
26         d[chr(i+c)] = chr((i+13) % 26 + c)
27
28 print("".join([d.get(c, c) for c in s]))
```

<https://github.com/python/cpython/blob/main/Lib/this.py>



```
1
2 import webbrowser
3 import hashlib
4
5 webbrowser.open("https://xkcd.com/353/")
6
7 def geohash(latitude, longitude, datedow):
8     '''Compute geohash() using the Munroe algorithm.
9
10    >>> geohash(37.421542, -122.085589, b'2005-05-26-10458.68')
11    37.857713 -122.544543
12
13    ...
14
15    # https://xkcd.com/426/
16    h = hashlib.md5(datedow, usedforsecurity=False).hexdigest()
17    p, q = [('%f' % float.fromhex('0.' + x)) for x in (h[:16], h[16:32])]
18    print('%d%s %d%s' % (latitude, p[1:], longitude, q[1:]))
```

<https://github.com/python/cpython/blob/main/Lib/antigravity.py>

Ok, pěkný vtip/reference a ničemu to nevadí, right? Right? Well...

Ok, pěkný vtip/reference a ničemu to nevadí, right? Right? Well...

webbrowser je... jak to říct... mocný.

Ok, pěkný vtip/reference a ničemu to nevadí, right? Right? Well...

webbrowser je... jak to říct... mocný.

```
$ export BROWSER="bash -c 'cat /etc/passwd' # %s"
$ python3 -c 'import antigravity'
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
# [...]
```

PODIVNÝ assert

Dokážme Python přimět aby dokázal následující?

```
1 print("Důkaz že '\"Smysl života' == 42' ", end="", flush=True)
2 try:
3     assert "Smysl života" == 42
4 except AssertionError:
5     print("se nepovedl :(")
6 else:
7     print("je úspěšný!")
```

-0

Remove assert statements and any code conditional on the value of `__debug__`. Augment the filename for compiled ([bytecode](#)) files by adding `.opt-1` before the `.pyc` extension (see [PEP 488](#)). See also [PYTHONOPTIMIZE](#).

| Changed in version 3.5: Modify `.pyc` filenames according to [PEP 488](#).

-00

Do `-0` and also discard docstrings. Augment the filename for compiled ([bytecode](#)) files by adding `.opt-2` before the `.pyc` extension (see [PEP 488](#)).

| Changed in version 3.5: Modify `.pyc` filenames according to [PEP 488](#).

<https://docs.python.org/3/using/cmdline.html#cmdoption-O>

Stačí tedy Python spustit s -O:

```
$ python3 -O script.py  
Důkaz že '"Smysl života' == 42' je úspěšný!
```

Stačí tedy Python spustit s -O:

```
$ python3 -O script.py  
Důkaz že '"Smysl života" == 42' je úspěšný!
```

Pokud bychom to chtěli "opravit":

```
1 print("Důkaz že '\\"Smysl života\\" == 42' ", end="", flush=True)  
2 try:  
3     if not ("Smysl života" == 42): raise AssertionError  
4 except AssertionError: print("se nepovedl :(")  
5 else: print("je úspěšný!")
```

SYNTAX

... a co vše dovolí.

Aneb náhodné věci, pro které jsem nechtěl dělat speciální sekci.

Pozor, tato část může zanechat závažné následky na vašem
mentálním zdraví.

Dokážeme napsat každý Python program jako oneliner?

`eval()` a `exec()` nepočítám, to už je podstatně odlišná věc.



stepech 21:08

Tak když dokážeš libovolný program v Pythonu napsat jako one liner....



Dokážeme napsat každý Python program jako oneliner?

`eval()` a `exec()` nepočítám, to už je podstatně odlišná věc.



stepech 21:08

Tak když dokážeš libovolný program v Pythonu napsat jako one liner....



Minimálně já teda nevím o způsobu jak napsat
`try: ... except: ...` na jeden řádek...

Kdo říká, že `class` má mít jen definice a assignments?

```
1 ADD_METHOD = "abc" in __import__("sys").argv
2 class Cls:
3     for x in range(3): print("Hello")
4     y = input("y: ")
5
6     if ADD_METHOD:
7         def method(self):
8             return "Hello from method"
9
10    cls = Cls()
11    print(cls.x)
12    print(cls.y)
13    try:
14        print(cls.method())
15    except AttributeError:
16        print("Method is missing")
```

class je vlastně jen "scope navíc"...

```
1 x = 777
2
3 class _:
4     x = 3
5     y = x*7
6     class _:
7         x = 666
8         y = 0
9     print(y*2) # 42
10
11 print(x) # 777
```

Téměř vše dokážeme napsat jen jako jeden expression:

```
1 [  
2     loop := [0],  
3     *[ [  
4         num := input("Number? "),  
5             [num := int(num)]  
6         if num.isdigit() else  
7             [  
8                 print("Not a number"),  
9                 loop.append(0)  
10            ]]  
11    ] for _ in loop),  
12        [msg := "That is a big number"]  
13    if num >= 5 else  
14        [msg := "That is a small number"],  
15    *[ [msg := msg + "."] for _ in range(num)],  
16    print(msg)  
17 ]
```

Téměř vše dokážeme napsat jen jako jeden expression:

```
1 [  
2     loop := [0],  
3     *[ [  
4         num := input("Number? "),  
5             [num := int(num)]  
6         if num.isdigit() else  
7             [  
8                 print("Not a number"),  
9                 loop.append(0)  
10            ]]  
11    ] for _ in loop),  
12        [msg := "That is a big number"]  
13    if num >= 5 else  
14        [msg := "That is a small number"],  
15    *[ [msg := msg + "."] for _ in range(num)],  
16    print(msg)  
17 ]
```

Téměř vše dokážeme napsat jen jako jeden expression:

```
1 [  
2     loop := [0],  
3     *[ [  
4         num := input("Number? "),  
5             [num := int(num)]  
6             if num.isdigit() else  
7                 [  
8                     print("Not a number"),  
9                     loop.append(0)  
10                ]]  
11    ] for _ in loop),  
12        [msg := "That is a big number"]  
13    if num >= 5 else  
14        [msg := "That is a small number"],  
15        *[ [msg := msg + "."] for _ in range(num)],  
16        print(msg)  
17 ]
```

Téměř vše dokážeme napsat jen jako jeden expression:

```
1 [  
2     loop := [0],  
3     *[ [  
4         num := input("Number? "),  
5             [num := int(num)]  
6         if num.isdigit() else  
7             [  
8                 print("Not a number"),  
9                 loop.append(0)  
10            ]]  
11    ] for _ in loop),  
12        [msg := "That is a big number"]  
13    if num >= 5 else  
14        [msg := "That is a small number"],  
15    *[ [msg := msg + "."] for _ in range(num)],  
16    print(msg)  
17 ]
```

Téměř vše dokážeme napsat jen jako jeden expression:

```
1 [  
2     loop := [0],  
3     *[ [  
4         num := input("Number? "),  
5             [num := int(num)]  
6             if num.isdigit() else  
7                 [  
8                     print("Not a number"),  
9                     loop.append(0)  
10                ]]  
11    ] for _ in loop),  
12        [msg := "That is a big number"]  
13    if num >= 5 else  
14        [msg := "That is a small number"],  
15    *[ [msg := msg + "."] for _ in range(num)],  
16    print(msg)  
17 ]
```

Co následující řádky? Které z nich jsou OK, které vyústí syntax error
a které skončí jinou výjimkou?

```
1 False .o: True
2 True[""]: 42
3 print.non_existing_attr: ...
4 _:{_:=""}=""
```

Co následující řádky? Které z nich jsou OK, které vyústí syntax error a které skončí jinou výjimkou?

```
1 False .o: True
2 True[""]: 42
3 print.non_existing_attr: ...
4 _:{_:=""}=""
```

Vše je validní syntaxe a dokonce je vše spustitelné bez výjimek.

```
$ echo 'True['""]: 42' | python3 -m dis
 0      RESUME          0

 1      SETUP_ANNOTATIONS
        RETURN_CONST      1 (None)
$ echo 'print.non_existing_attr: ...' | python3 -m dis
 0      RESUME          0

 1      SETUP_ANNOTATIONS
        LOAD_NAME         0 (print)
        POP_TOP
        RETURN_CONST     1 (None)
```

- <https://peps.python.org/pep-0526/#annotating-expressions> ("simple" vs "not simple" names/annotations)
- <https://github.com/python/cpython/blob/main/Python/codegen.c#L5383> (`int codegen_annassign(...)`)

for umí dosazovat nejen do jednoduché proměnné...

```
1 o = type("",(),{})()
2 for o.abc in range(10):
3     pass
4 print(o.abc) # 9
5
6 l = [None]
7 for l[0] if l[0] else -1] in range(43):
8     pass
9 print(l) # [42]
```

Jednoduché dosazování se přece nemůže být tak divné, či?

```
>>> a=a [θ]=[θ]
```

Jednoduché dosazování se přece nemůže být tak divné, či?

```
>>> a=a [θ]=[θ]  
>>> a  
[ ... ]
```

Jednoduché dosazování se přece nemůže být tak divné, či?

```
>>> a=a [0]=[0]
>>> a
[...]
>>> a[0][0][0]
[...]
```

Jednoduché dosazování se přece nemůže být tak divné, či?

```
>>> a=a [0]=[0]
>>> a
[...]
>>> a[0][0][0]
[...]
>>> a is a[0]
True
```

ANNOTATIONS

Typové anotace jsou skvělé, všichni je milujeme. Co takhle následující kód?

```
1 class A:  
2     def f(self) -> B:  
3         ...  
4  
5 class B:  
6     def f(self) -> A:  
7         ...
```

ANNOTATIONS

Typové anotace jsou skvělé, všichni je milujeme. Co takhle následující kód?

```
1 class A:  
2     def f(self) -> B:  
3         ...  
4  
5 class B:  
6     def f(self) -> A:  
7         ...
```

NameError: name 'B' is not defined

Anotaci lze zapsat jako string...

```
1 class A:  
2     def f(self) -> "B":  
3         ...  
4  
5 class B:  
6     def f(self) -> A:  
7         ...
```

... to je ale takové nepěkné...

Lze použít `from __future__ import annotations`

```
1 from __future__ import annotations
2
3 class A:
4     def f(self) -> B:
5         ...
6
7 class B:
8     def f(self) -> A:
9         ...
```

Lze použít `from __future__ import annotations`

```
1 from __future__ import annotations
2
3 class A:
4     def f(self) -> B:
5         ...
6
7 class B:
8     def f(self) -> A:
9         ...
```

Ale co se to vlastně stalo?

```
from __future__ import annotations
```

převede anotace do stringů. Mělo být "by default" od Pythonu 3.10, ale ...

Nyní to vypadá, že se problém bude řešit jinak (PEP 649).

- PEP 563 (Postponed Evaluation of Annotations):
<https://peps.python.org/pep-0563/>
- PEP 649 (Deferred Evaluation Of Annotations Using Descriptors):
<https://peps.python.org/pep-0649/>

EXCEPTIONS

Dejme tomu, že píšeme nějakou aplikaci...

main.py

```
1 class AppError(Exception):
2     pass
3
4 def check(x):
5     if x == 555:
6         raise AppError("Number is not allowed")
7
8 if __name__ == "__main__":
9     x = int(input("Number: "))
10
11 try:
12     check(x)
13 except AppError as e:
14     print(f"Validation error ({e})")
15     exit(1)
16
17 print("All good")
```

EXCEPTIONS

Dejme tomu, že píšeme nějakou aplikaci...

main.py

```
1 class AppError(Exception):
2     pass
3
4 if __name__ == "__main__":
5     import validator
6
7     x = int(input("Number: "))
8
9     try:
10         validator.check(x)
11     except AppError as e:
12         print(f"Validation error ({e})")
13         exit(1)
14
15     print("All good")
```

validator.py

```
1 from main import AppError
2
3 def check(x):
4     if x == 555:
5         raise AppError("Number is not allowed")
```

Po separování validační funkce do samostatného souboru se to nechová tak jak bychom čekali... Co s tím?

Po separování validační funkce do samostatného souboru se to nechová tak jak bychom čekali... Co s tím?

<class 'main.AppError'> vs <class '__main__.AppError'>

Po separování validační funkce do samostatného souboru se to nechová tak jak bychom čekali... Co s tím?

<class 'main.AppError'> vs <class '__main__.AppError'>

validator.py importuje main a ne __main__, tzn. main.py se načte jako nový modul (a tedy se vytvoří i nová třída).

Po separování validační funkce do samostatného souboru se to nechová tak jak bychom čekali... Co s tím?

<class 'main.AppError'> vs <class '__main__.AppError'>

validator.py importuje main a ne __main__, tzn. main.py se načte jako nový modul (a tedy se vytvoří i nová třída).

Fix? Nic moc udělat nelze, prakticky "as intended"... Správné řešení je kód přepsat (př. přemístit AppError do jiného souboru).

NUMBER IS NUMBER

Výstup následujícího kódu?

```
1 print(0 is 0)
2 print(0 is 1)
3 print(1 is 1)
4 print(255 is 255)
5 print(256 is 256)
6 print(257 is 257)
7 print(9876543210 is 9876543210)
```

NUMBER IS NUMBER

Výstup následujícího kódu?

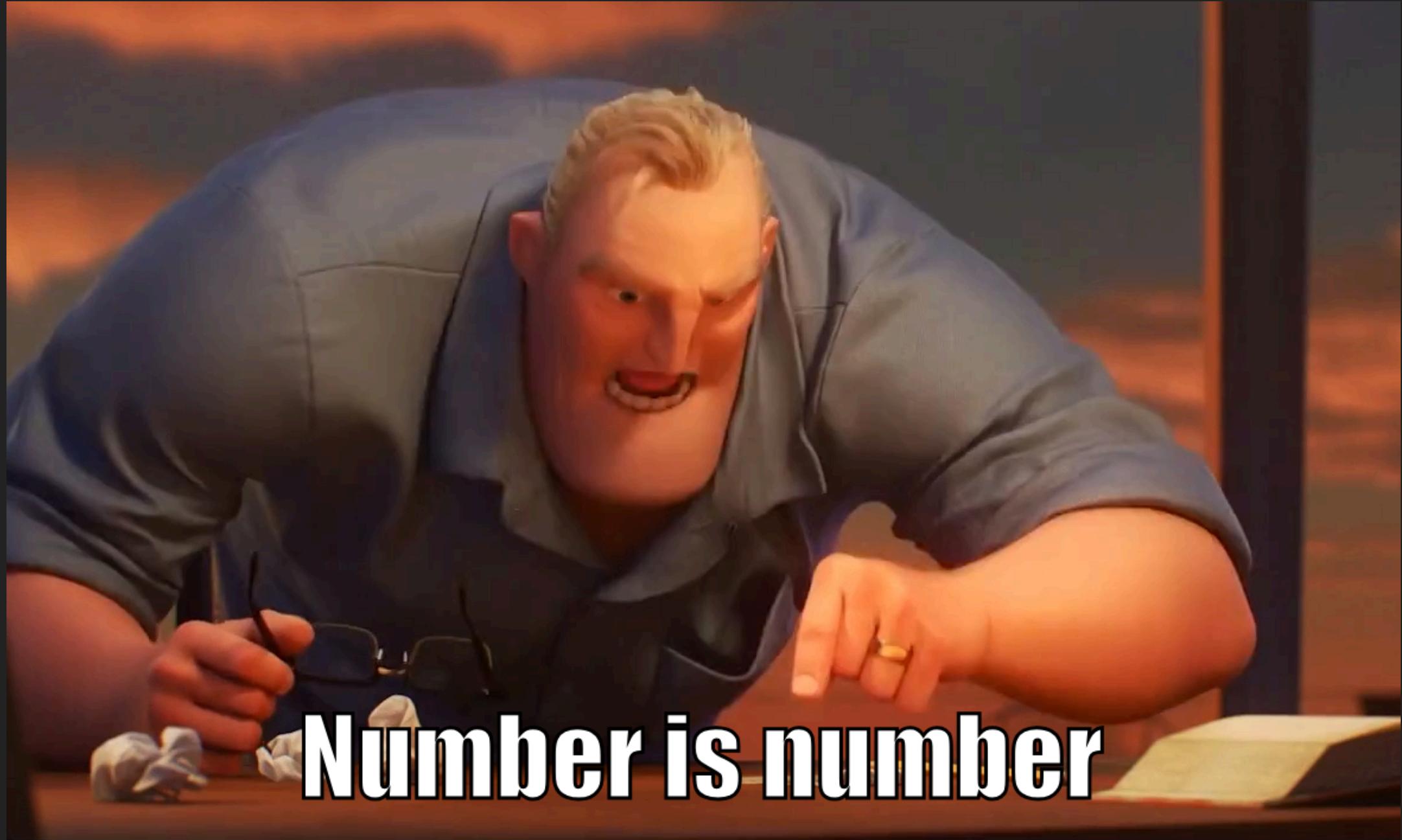
```
1 print(0 is 0) # True
2 print(0 is 1) # False
3 print(1 is 1) # True
4 print(255 is 255) # True
5 print(256 is 256) # True
6 print(257 is 257) # True
7 print(9876543210 is 9876543210) # True
```

NUMBER IS NUMBER

Výstup následujícího kódu?

```
1 print(0 is 0) # True
2 print(0 is 1) # False
3 print(1 is 1) # True
4 print(255 is 255) # True
5 print(256 is 256) # True
6 print(257 is 257) # True
7 print(9876543210 is 9876543210) # True
```

SyntaxWarning: "is" with 'int' literal. Did you mean "=="?



Number is number

```
1 x = 256
2 y = 256
3 print(x is y)
4 print(255 + 1 is 256)
5 one = 1
6 print(255 + one is 256)
```

```
1 x = 256
2 y = 256
3 print(x is y) # True
4 print(255 + 1 is 256) # True
5 one = 1
6 print(255 + one is 256) # True
```

```
1 x = 256
2 y = 256
3 print(x is y) # True
4 print(255 + 1 is 256) # True
5 one = 1
6 print(255 + one is 256) # True
```

```
1 x = 257
2 y = 257
3 print(x is y)
4 print(256 + 1 is 257)
5 one = 1
6 print(256 + one is 257)
```

```
1 x = 256
2 y = 256
3 print(x is y) # True
4 print(255 + 1 is 256) # True
5 one = 1
6 print(255 + one is 256) # True
```

```
1 x = 257
2 y = 257
3 print(x is y) # True
4 print(256 + 1 is 257) # True
5 one = 1
6 print(256 + one is 257) # False !!!
```

Implementace každé operace končí voláním `maybe_small_long()`.

```
58 static PyObject *
59 get_small_int(sdigit ival)
60 {
61     assert(IS_SMALL_INT(ival));
62     return (PyObject *)&_PyLong_SMALL_INTS[_PY_NSMALLNEGINTS + ival];
63 }
64
65 static PyLongObject *
66 maybe_small_long(PyLongObject *v)
67 {
68     if (v && _PyLong_IsCompact(v)) {
69         stwodigits ival = medium_value(v);
70         if (IS_SMALL_INT(ival)) {
71             _Py_DECREF_INT(v);
72             return (PyLongObject *)get_small_int((sdigit)ival);
73         }
74     }
75     return v;
76 }
```

Implementace každé operace končí voláním `maybe_small_long()`.

```
58 static PyObject *
59 get_small_int(sdigit ival)
60 {
61     assert(IS_SMALL_INT(ival));
62     return (PyObject *)&_PyLong_SMALL_INTS[_PY_NSMALLNEGINTS + ival];
63 }
64
65 static PyLongObject *
66 maybe_small_long(PyLongObject *v)
67 {
68     if (v && _PyLong_IsCompact(v)) {
69         stwodigits ival = medium_value(v);
70         if (IS_SMALL_INT(ival)) {
71             _Py_DECREF_INT(v);
72             return (PyLongObject *)get_small_int((sdigit)ival);
73         }
74     }
75     return v;
76 }
```

Implementace každé operace končí voláním `maybe_small_long()`.

```
58 static PyObject *
59 get_small_int(sdigit ival)
60 {
61     assert(IS_SMALL_INT(ival));
62     return (PyObject *)&_PyLong_SMALL_INTS[_PY_NSMALLNEGINTS + ival];
63 }
64
65 static PyLongObject *
66 maybe_small_long(PyLongObject *v)
67 {
68     if (v && _PyLong_IsCompact(v)) {
69         stwodigits ival = medium_value(v);
70         if (IS_SMALL_INT(ival)) {
71             _Py_DECREF_INT(v);
72             return (PyLongObject *)get_small_int((sdigit)ival);
73         }
74     }
75     return v;
76 }
```

Implementace každé operace končí voláním `maybe_small_long()`.

```
58 static PyObject *
59 get_small_int(sdigit ival)
60 {
61     assert(IS_SMALL_INT(ival));
62     return (PyObject *)&_PyLong_SMALL_INTS[_PY_NSMALLNEGINTS + ival];
63 }
64
65 static PyLongObject *
66 maybe_small_long(PyLongObject *v)
67 {
68     if (v && _PyLong_IsCompact(v)) {
69         stwodigits ival = medium_value(v);
70         if (IS_SMALL_INT(ival)) {
71             _Py_DECREF_INT(v);
72             return (PyLongObject *)get_small_int((sdigit)ival);
73         }
74     }
75     return v;
76 }
```

Implementace každé operace končí voláním `maybe_small_long()`.

```
58 static PyObject *
59 get_small_int(sdigit ival)
60 {
61     assert(IS_SMALL_INT(ival));
62     return (PyObject *)&_PyLong_SMALL_INTS[_PY_NSMALLNEGINTS + ival];
63 }
64
65 static PyLongObject *
66 maybe_small_long(PyLongObject *v)
67 {
68     if (v && _PyLong_IsCompact(v)) {
69         stwodigits ival = medium_value(v);
70         if (IS_SMALL_INT(ival)) {
71             _Py_DECREF_INT(v);
72             return (PyLongObject *)get_small_int((sdigit)ival);
73         }
74     }
75     return v;
76 }
```


Ale proč tedy platí $256+1$ is 257 ?

Ale proč tedy platí $256+1$ is 257?

1. Kompilátor optimalizuje ($\Rightarrow 256+1 \Rightarrow 257$)
2. Kompilace má "constant cache"

Ale proč tedy platí **256+1 is 257?**

1. Kompilátor optimalizuje ($\Rightarrow 256+1 \Rightarrow 257$)
2. Kompilace má "constant cache"

```
$ echo -e "256+1 is 257" | python3 -m dis
 0      RESUME          0
 1      LOAD_CONST      0 (257)
        LOAD_CONST      0 (257)
        IS_OP            0
        POP_TOP
        RETURN_CONST    1 (None)
```

Ale proč tedy platí **256+1 is 257?**

1. Kompilátor optimalizuje ($\Rightarrow 256+1 \Rightarrow 257$)
2. Kompilace má "constant cache"

```
$ echo -e "256+1 is 257" | python3 -m dis
 0      RESUME          0
 1      LOAD_CONST      0 (257)
        LOAD_CONST      0 (257)
        IS_OP            0
        POP_TOP
        RETURN_CONST     1 (None)
```

```
>>> code = compile("257 is 257", "", "exec")
>>> code.co_consts
(257, None)
```

Co takhle interaktivní režim?

```
>>> 257 is 257
# ???
>>> x = 257
>>> y = 257
>>> x is y
# ???
```

Co takhle interaktivní režim?

```
>>> 257 is 257
True
>>> x = 257
>>> y = 257
>>> x is y
False
```

Co takhle interaktivní režim?

```
>>> 257 is 257
True
>>> x = 257
>>> y = 257
>>> x is y
False
```

Interaktivní režim se "kompiluje" řádek po řádku. Constant cache tedy moc nefunguje, protože vše se kompiluje zvlášť.

KOMENTÁŘE

KOMENTÁŘE SE ŠPETKOU MAGIE

KOMENTÁŘE SE ŠPETKOU MAGIE

Udělá následující kód něco?

```
1 # This is normal Python comment, nothing extra...
2 # But what about following line, will it do something?
3 # \u000Aprint("Hello Wolrd!")
```

KOMENTÁŘE SE ŠPETKOU MAGIE

Udělá následující kód něco?

```
1 # This is normal Python comment, nothing extra...
2 # But what about following line, will it do something?
3 # \u000Aprint("Hello Wolrd!")
```

Samozřejmě že ne, vždyť je to komentář!

KOMENTÁŘE SE ŠPETKOU MAGIE

Udělá následující kód něco?

```
1 # This is normal Python comment, nothing extra...
2 # But what about following line, will it do something?
3 # \u000Aprint("Hello Wolrd!")
```

Samozřejmě že ne, vždyť je to komentář!

Na druhou stranu se to ale objevilo v tomto talku...

Nestačilo by Python prostě jen poprosit?

Nestačilo by Python prostě jen poprosit?

```
1 # Python, pretty please.  
2 # Would you be so nice and use following encoding: unicode_escape?  
3 # Thank you very much!  
4 # \u000Aprint("Hello Wolrd!")
```

Nestačilo by Python prostě jen poprosit?

```
1 # Python, pretty please.  
2 # Would you be so nice and use following encoding: unicode_escape?  
3 # Thank you very much!  
4 # \u000Aprint("Hello Wolrd!")
```

```
$ python3 pretty_please.py  
Hello Wolrd!
```

Nestačilo by Python prostě jen poprosit?

```
1 # Python, pretty please.  
2 # Would you be so nice and use following encoding: unicode_escape?  
3 # Thank you very much!  
4 # \u000Aprint("Hello Wolrd!")
```

```
$ python3 pretty_please.py  
Hello Wolrd!
```

Přesně jak mi rodiče říkali - stačí použít jen "magické slovíčko".

2.1.4. Encoding declarations

If a comment in the first or second line of the Python script matches the regular expression `coding[=:]\s*([-\\w.]+)`, this comment is processed as an encoding declaration; the first group of this expression names the encoding of the source code file. The encoding declaration must appear on a line of its own. If it is the second line, the first line must also be a comment-only line. The recommended forms of an encoding expression are

```
# -*- coding: <encoding-name> -*-
```

which is recognized also by GNU Emacs, and

```
# vim:fileencoding=<encoding-name>
```

which is recognized by Bram Moolenaar's VIM.

If no encoding declaration is found, the default encoding is UTF-8. If the implicit or explicit encoding of a file is UTF-8, an initial UTF-8 byte-order mark (`b'\xef\xbb\xbf'`) is ignored rather than being a syntax error.

If an encoding is declared, the encoding name must be recognized by Python (see [Standard Encodings](#)). The encoding is used for all lexical analysis, including string literals, comments and identifiers.

Smutý fakt je, že se musí jednat o "text encoding", takže třeba
`uu_codec` nebo `hex_codec` nejde... :(

Teoreticky lze použít EBCDIC encodingy, ale prakticky ne (AFAIK
nelze udělat validní syntax z konvertovaného komentáře)

Smutý fakt je, že se musí jednat o "text encoding", takže třeba
`uu_codec` nebo `hex_codec` nejde... :(

Teoreticky lze použít EBCDIC encodingy, ale prakticky ne (AFAIK
nelze udělat validní syntax z konvertovaného komentáře)

Ale tohle je třeba validní (a spustitelný) Python kód:

```
1 # encoding: hz
2 # Python is just great~
3 $"$ So many possibilities to mess up the source code! $$$ <3
4 print("Hello World!")
```

HASHOVÁNÍ

hash() zahashuje daný objekt...

```
>>> hash("Whatever")  
1234019493422677444
```

HASHOVÁNÍ

hash() zahashuje daný objekt...

```
>>> hash("Whatever")
1234019493422677444
>>> hash((1,2,3))
529344067295497451
```

HASHOVÁNÍ

hash() zahashuje daný objekt...

```
>>> hash("Whatever")
1234019493422677444
>>> hash((1,2,3))
529344067295497451
>>> hash({})
TypeError: unhashable type: 'dict'
```

HASHOVÁNÍ

hash() zahashuje daný objekt...

```
>>> hash("Whatever")
1234019493422677444
>>> hash((1,2,3))
529344067295497451
>>> hash({})
TypeError: unhashable type: 'dict'
>>> hash(42)
42
```

Zaměříme se na čísla:

```
>>> hash(42)
42
>>> hash(-42)
-42
```

Zaměříme se na čísla:

```
>>> hash(42)
42
>>> hash(-42)
-42
>>> [hash(x) for x in range(10)]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Zaměříme se na čísla:

```
>>> hash(42)
42
>>> hash(-42)
-42
>>> [hash(x) for x in range(10)]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> [hash(x) for x in range(0,-10,-1)]
[0, -2, -2, -3, -4, -5, -6, -7, -8, -9]
```

Vše v pořádku...

Zaměříme se na čísla:

```
>>> hash(42)
42
>>> hash(-42)
-42
>>> [hash(x) for x in range(10)]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> [hash(x) for x in range(0,-10,-1)]
[0, -2, -2, -3, -4, -5, -6, -7, -8, -9]
```

Ou, vlastně ne. Asi to mám nějaký rozbitý...

Zaměříme se na čísla:

```
>>> hash(42)
42
>>> hash(-42)
-42
>>> [hash(x) for x in range(10)]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> [hash(x) for x in range(0,-10,-1)]
[0, -2, -2, -3, -4, -5, -6, -7, -8, -9]
```

Ou, vlastně ne. Asi to mám nějaký rozbitý...

```
>>> hash(-1)
-2
```

Ok, tak u čísel je to nějaké divné... Co vlastní objekty?

```
1 class A:  
2     def __init__(self, x):  
3         self.x = x  
4     def __hash__(self):  
5         return self.x
```

Ok, tak u čísel je to nějaké divné... Co vlastní objekty?

```
1 class A:  
2     def __init__(self, x):  
3         self.x = x  
4     def __hash__(self):  
5         return self.x
```

```
>>> hash(A(100))  
100
```

Ok, tak u čísel je to nějaké divné... Co vlastní objekty?

```
1 class A:  
2     def __init__(self, x):  
3         self.x = x  
4     def __hash__(self):  
5         return self.x
```

```
>>> hash(A(100))  
100  
>>> hash(A(-123))  
-123
```

Ok, tak u čísel je to nějaké divné... Co vlastní objekty?

```
1 class A:  
2     def __init__(self, x):  
3         self.x = x  
4     def __hash__(self):  
5         return self.x
```

```
>>> hash(A(100))  
100  
>>> hash(A(-123))  
-123  
>>> hash(A(-1))  
-2
```

Ok, tak u čísel je to nějaké divné... Co vlastní objekty?

```
1 class A:  
2     def __init__(self, x):  
3         self.x = x  
4     def __hash__(self):  
5         return self.x
```

```
>>> hash(A(100))  
100  
>>> hash(A(-123))  
-123  
>>> hash(A(-1))  
-2  
>>> A(-1).__hash__()  
-1
```

Je to nějaké podivné, určitě s explicitní **-1** to bude v pořádku...

Je to nějaké podivné, určitě s explicitní **-1** to bude v pořádku...

```
1 class A:  
2     def __hash__(self):  
3         return -1  
4  
5 print(hash(A())) # => -2
```

Je to nějaké podivné, určitě s explicitní **-1** to bude v pořádku...

```
1 class A:  
2     def __hash__(self):  
3         return -1  
4  
5 print(hash(A())) # => -2
```

Co se to děje? Nápady?

Je to nějaké podivné, určitě s explicitní **-1** to bude v pořádku...

```
1 class A:  
2     def __hash__(self):  
3         return -1  
4  
5 print(hash(A())) # => -2
```

Co se to děje? Nápady?

- Je to z nějakého důvodu bugnuté

Je to nějaké podivné, určitě s explicitní **-1** to bude v pořádku...

```
1 class A:  
2     def __hash__(self):  
3         return -1  
4  
5 print(hash(A())) # => -2
```

Co se to děje? Nápady?

- Je to z nějakého důvodu bugnuté
- Někde je něco ve stylu **if (output == -1) {return -2;}**

Je to nějaké podivné, určitě s explicitní **-1** to bude v pořádku...

```
1 class A:  
2     def __hash__(self):  
3         return -1  
4  
5 print(hash(A())) # => -2
```

Co se to děje? Nápady?

- Je to z nějakého důvodu bugnuté
- Někde je něco ve stylu **if (output == -1) {return -2;}**
- Dělám si z vás strandu a nějak jsem patchnul hash()

Je to nějaké podivné, určitě s explicitní **-1** to bude v pořádku...

```
1 class A:  
2     def __hash__(self):  
3         return -1  
4  
5 print(hash(A())) # => -2
```

Co se to děje? Nápady?

- Je to z nějakého důvodu bugnuté
- Někde je něco ve stylu `if (output == -1) {return -2;}`
- Dělám si z vás strandu a nějak jsem patchnul `hash()`
- Prostě protože proto...

Je to nějaké podivné, určitě s explicitní **-1** to bude v pořádku...

```
1 class A:  
2     def __hash__(self):  
3         return -1  
4  
5 print(hash(A())) # => -2
```

Co se to děje? Nápady?

- Je to z nějakého důvodu bugnuté
- Někde je něco ve stylu `if (output == -1) {return -2;}`
- Dělá m si z vás srandu a nějak jsem patchnul `hash()`
- Prostě protože proto...

Víte jak se v C tradičně signalizují chyby? :)

Víte jak se v C tradičně signalizují chyby? :)

Ano, přesně tak...

```
132 Py_hash_t
133 Py_HashPointer(const void *ptr)
134 {
135     Py_hash_t hash = _Py_HashPointerRaw(ptr);
136     if (hash == -1) {
137         hash = -2;
138     }
139     return hash;
140 }
141
142 Py_hash_t
143 PyObject_GenericHash(PyObject *obj)
144 {
145     return Py_HashPointer(obj);
146 }
```

<https://github.com/python/cpython/blob/main/Python/pyhash.c#L132-L146>

```
10091 static Py_hash_t
10092 slot_tp_hash(PyObject *self)
10093 {
10094     PyObject *res;
10095     int attr_is_none = 0;
10096     res = maybe_call_special_no_args(self, &_Py_ID(__hash__), &attr_is_none);
10097     if (attr_is_none || res == NULL) {
10098         if (PyErr_Occurred()) {
10099             return -1;
10100         }
10101         return PyObject_HashNotImplemented(self);
10102     }
10103     /* ... */
10104     /* -1 is reserved for errors. */
10105     if (h == -1)
10106         h = -2;
10107     Py_DECREF(res);
10108     return h;
10109 }
```

<https://github.com/python/cpython/blob/main/Objects/typeobject.c#L10091-L10126>

```
3666 static Py_hash_t
3667 long_hash(PyObject *obj)
3668 {
3669     PyLongObject *v = (PyLongObject *)obj;
3670     Py_uhash_t x;
3671     Py_ssize_t i;
3672     int sign;
3673
3674     if (_PyLong_IsCompact(v)) {
3675         x = (Py_uhash_t)_PyLong_CompactValue(v);
3676         if (x == (Py_uhash_t)-1) {
3677             x = (Py_uhash_t)-2;
3678         }
3679         return x;
3680     }
3681     /* ... */
3682     if (x == (Py_uhash_t)-1)
3683         x = (Py_uhash_t)-2;
3684     return (Py_hash_t)x;
3685 }
```

<https://github.com/python/cpython/blob/main/Objects/longobject.c#L3666-L3720>

METODY

Jednoduché printování argumentů:

```
1 class A:  
2     def method(self, *a):  
3         print(*a)
```

```
>>> a = A()  
>>> a.method(123, "something")  
# ???
```

METODY

Jednoduché printování argumentů:

```
1 class A:  
2     def method(self, *a):  
3         print(*a)
```

```
>>> a = A()  
>>> a.method(123, "something")  
123 something
```

Vše v normálu, trochu to pozměníme...

```
1 class A:  
2     def method(*a):  
3         print(*a)
```

```
>>> a = A()  
>>> a.method(123, "something")  
# ???
```

Vše v normálu, trochu to pozměníme...

```
1 class A:  
2     def method(*a):  
3         print(*a)
```

```
>>> a = A()  
>>> a.method(123, "something")  
<__main__.A object at 0x000001F679167230> 123 something
```

Ted' ale to můžeme napsat ještě úsporněji:

```
1 class A:  
2     method = print
```

```
>>> a = A()  
>>> a.method(123, "something")  
# ???
```

Ted' ale to můžeme napsat ještě úsporněji:

```
1 class A:  
2     method = print
```

```
>>> a = A()  
>>> a.method(123, "something")  
123 something
```

Ted' ale to můžeme napsat ještě úsporněji:

```
1 class A:  
2     method = print
```

```
>>> a = A()  
>>> a.method(123, "something")  
123 something
```

Hmm... To je nějaké zvláštní...

Ok, tak to přímé dosazení asi má nějaký vedlejší efekt... Zkusme dosadit nějakou vlastní funkci:

```
1 wrapper = lambda *a: print(*a)
2 class A:
3     method = wrapper
```

```
>>> a = A()
>>> a.method(123, "something")
# ???
```

Ok, tak to přímé dosazení asi má nějaký vedlejší efekt... Zkusme dosadit nějakou vlastní funkci:

```
1 wrapper = lambda *a: print(*a)
2 class A:
3     method = wrapper
```

```
>>> a = A()
>>> a.method(123, "something")
<__main__.A object at 0x0000010A032C7230> 123 something
```

Ok, tak to přímé dosazení asi má nějaký vedlejší efekt... Zkusme dosadit nějakou vlastní funkci:

```
1 wrapper = lambda *a: print(*a)
2 class A:
3     method = wrapper
```

```
>>> a = A()
>>> a.method(123, "something")
<__main__.A object at 0x0000010A032C7230> 123 something
```

Wat?

Builtin "funkce" jsou ve skutečnosti metody (jen nepoužívají `self`).

```
928 #define BUILTIN_PRINT_METHODDEF      \
929     {"print", _PyCFunction_CAST(builtin_print), METH_FASTCALL|METH_KEYWORDS,
930      builtin_print__doc__},

3271 static PyMethodDef builtin_methods[] = {
3272     {"__build_class__", _PyCFunction_CAST(builtin__build_class__),
3273      METH_FASTCALL | METH_KEYWORDS, build_class_doc},
3274     BUILTIN__IMPORT__METHODDEF
3275     /* ... */
3276     BUILTIN_POW_METHODDEF
3277     BUILTIN_PRINT_METHODDEF
3278     BUILTIN_REPR_METHODDEF
3279     BUILTIN_ROUND_METHODDEF
3280     BUILTIN_SETATTR_METHODDEF
3281     BUILTIN_SORTED_METHODDEF
3282     BUILTIN_SUM_METHODDEF
3283     {"vars",             builtin_vars,      METH_VARARGS, vars_doc},
3284     {NULL,               NULL},
```

- <https://github.com/python/cpython/blob/main/Python/clinic/bltinmodule.c.h#L928-L929>
- <https://github.com/python/cpython/blob/main/Python/bltinmodule.c#L3271-L3318>

Builtin "funkce" jsou ve skutečnosti metody (jen nepoužívají `self`).

```
928 #define BUILTIN_PRINT_METHODDEF      \
929     {"print", _PyCFunction_CAST(builtin_print), METH_FASTCALL|METH_KEYWORDS,
930      builtin_print__doc__},

3271 static PyMethodDef builtin_methods[] = {
3272     {"__build_class__", _PyCFunction_CAST(builtin__build_class__),
3273      METH_FASTCALL | METH_KEYWORDS, build_class_doc},
3274     BUILTIN__IMPORT__METHODDEF
3275     /* ... */
3276     BUILTIN_POW_METHODDEF
3277     BUILTIN_PRINT_METHODDEF
3278     BUILTIN_REPR_METHODDEF
3279     BUILTIN_ROUND_METHODDEF
3280     BUILTIN_SETATTR_METHODDEF
3281     BUILTIN_SORTED_METHODDEF
3282     BUILTIN_SUM_METHODDEF
3283     {"vars",           builtin_vars,      METH_VARARGS, vars_doc},
3284     {NULL,             NULL},
```

- <https://github.com/python/cpython/blob/main/Python/clinic/bltinmodule.c.h#L928-L929>
- <https://github.com/python/cpython/blob/main/Python/bltinmodule.c#L3271-L3318>

VOODOO / ČERNÁ MAGIE

Jak si změnit builtin typy?

VOODOO / ČERNÁ MAGIE

Jak si změnit builtin typy?

```
object.attr = "Hello World!"
```

```
TypeError: cannot set 'attr' attribute of immutable type 'object'
```

VOODOO / ČERNÁ MAGIE

Jak si změnit builtin typy?

```
setattr(object, "attr", 42)
```

```
TypeError: cannot set 'attr' attribute of immutable type 'object'
```

VOODOO / ČERNÁ MAGIE

Jak si změnit builtin typy?

```
object.__setattr__(object, "attr", 42)
```

```
TypeError: can't apply this __setattr__ to type object
```

VOODOO / ČERNÁ MAGIE

Jak si změnit builtin typy?

```
object.__dict__["attr"] = 42
```

```
TypeError: 'mappingproxy' object does not support item assignment
```

VOODOO / ČERNÁ MAGIE

Jak si změnit builtin typy?

```
object.__dict__["attr"] = 42
```

```
TypeError: 'mappingproxy' object does not support item assignment
```

mappingproxy je speciální read-only wrapper kolem nějakého mappingu. Neexistuje interface, jak se dostat k originálnímu mappingu, ale...

```
1232 static PyObject *
1233 mappingproxy_richcompare(PyObject *self, PyObject *w, int op)
1234 {
1235     mappingproxyobject *v = (mappingproxyobject *)self;
1236     return PyObject_RichCompare(v->mapping, w, op);
1237 }
```

<https://github.com/python/cpython/blob/main/Objects/descrobject.c#L1232-L1237>

```
1232 static PyObject *
1233 mappingproxy_richcompare(PyObject *self, PyObject *w, int op)
1234 {
1235     mappingproxyobject *v = (mappingproxyobject *)self;
1236     return PyObject_RichCompare(v->mapping, w, op);
1237 }
```

<https://github.com/python/cpython/blob/main/Objects/descrobject.c#L1232-L1237>

```
1232 static PyObject *
1233 mappingproxy_richcompare(PyObject *self, PyObject *w, int op)
1234 {
1235     mappingproxyobject *v = (mappingproxyobject *)self;
1236     return PyObject_RichCompare(v->mapping, w, op);
1237 }
```

<https://github.com/python/cpython/blob/main/Objects/descrobject.c#L1232-L1237>

Do `PyObject_RichCompare()` se předává `v->mapping`, tedy onen původní `mapping`. Takže...

Když si vytvoříme vlastní typ, vracející z porovnávání "druhý objekt", tak co dostaneme zpátky?

```
1 class A:  
2     def __eq__(self, other):  
3         return other
```

Když si vytvoříme vlastní typ, vracející z porovnávání "druhý objekt", tak co dostaneme zpátky?

```
1 class A:  
2     def __eq__(self, other):  
3         return other
```

```
>>> type(object.__dict__)  
<class 'mappingproxy'>  
>>> type(object.__dict__ == A())  
<class 'dict'>  
>>> object_mapping = object.__dict__ == A()
```

Nyní může začít zábava:

```
>>> object_mapping["attr"] = 42
>>> object.attr
42
```

Nyní může začít zábava:

```
>>> object_mapping["attr"] = 42
>>> object.attr
42
>>> object().attr
42
```

Nyní může začít zábava:

```
>>> object_mapping["attr"] = 42
>>> object.attr
42
>>> object().attr
42
>>> [].attr
42
```

Nyní může začít zábava:

```
>>> object_mapping["attr"] = 42
>>> object.attr
42
>>> object().attr
42
>>> [].attr
42
>>> object_mapping["func"] = lambda *a,**k: print(a, k)
>>> None.func()
(None,)
```

Nyní může začít zábava:

```
>>> object_mapping["attr"] = 42
>>> object.attr
42
>>> object().attr
42
>>> [].attr
42
>>> object_mapping["func"] = lambda *a,**k: print(a, k)
>>> None.func()
(None,)
>>> (dict.__dict__ == A())["get"] = lambda *a,**k: 10
Segmentation fault (core dumped)
```

```
1 class A:  
2     def __eq__(self, other):  
3         return other  
4  
5 dict_mapping = dict.__dict__ == A()  
6  
7 print({}.get)  
8 dict_mapping["get"] = lambda *a,**k: ...  
9 print({}.get)
```

```
1 class A:  
2     def __eq__(self, other):  
3         return other  
4  
5 dict_mapping = dict.__dict__ == A()  
6  
7 print({}.get)  
8 dict_mapping["get"] = lambda *a,**k: ...  
9 print({}.get)
```

```
$ python3 script.py  
<built-in method get of dict object at 0x7efc2af50300>  
<built-in method write of _io.TextIOWrapper object at 0x7efc2b096500>
```

```
1 class A:  
2     def __eq__(self, other):  
3         return other  
4  
5 dict_mapping = dict.__dict__ == A()  
6  
7 print({}.get)  
8 dict_mapping["get"] = lambda *a,**k: ...  
9 print({}.get)
```

```
$ python3 script.py  
<built-in method get of dict object at 0x7efc2af50300>  
<built-in method write of _io.TextIOWrapper object at 0x7efc2b096500>
```

Kdo by to byl řekl, že se budou dít divné věci... :)

ZÁVĚR

Je ještě spousta věcí, co jsem přeskočil, ale ukázalo se, že nejsem schopný do jedné přednášky dát všechno.

ZÁVĚR

Je ještě spousta věcí, co jsem přeskočil, ale ukázalo se, že nejsem schopný do jedné přednášky dát všechno.

- pickle

ZÁVĚR

Je ještě spousta věcí, co jsem přeskočil, ale ukázalo se, že nejsem schopný do jedné přednášky dát všechno.

- pickle
- Typing (ještě horší věci s typovými anotacemi)

ZÁVĚR

Je ještě spousta věcí, co jsem přeskočil, ale ukázalo se, že nejsem schopný do jedné přednášky dát všechno.

- pickle
- Typing (ještě horší věci s typovými anotacemi)
- Low level stuff (viz třeba "Unsafe Python")

ZÁVĚR

Je ještě spousta věcí, co jsem přeskočil, ale ukázalo se, že nejsem schopný do jedné přednášky dát všechno.

- pickle
- Typing (ještě horší věci s typovými anotacemi)
- Low level stuff (viz třeba "Unsafe Python")
- Auditing

ZÁVĚR

Je ještě spousta věcí, co jsem přeskočil, ale ukázalo se, že nejsem schopný do jedné přednášky dát všechno.

- pickle
- Typing (ještě horší věci s typovými anotacemi)
- Low level stuff (viz třeba "Unsafe Python")
- Auditing
- Balíčky

ZÁVĚR

Je ještě spousta věcí, co jsem přeskočil, ale ukázalo se, že nejsem schopný do jedné přednášky dát všechno.

- pickle
- Typing (ještě horší věci s typovými anotacemi)
- Low level stuff (viz třeba "Unsafe Python")
- Auditing
- Balíčky
- ...

DÍKY ZA POZORNOST!

Otázky?

Případně moje Discord DMs (#Hackrrr) jsou vždy otevřené na jakoukoliv Python šílenost či PyJail...