

## Konteksti

Pythonov modul `contextlib` - ki ima, kot bi bilo bistremu človeku jasno že iz imena, nekaj opraviti s konteksti - ima funkcijo `chdir`, s katero lahko začasno zamenjamo trenutni delovni direktorij.

```
import os
import contextlib

print(os.getcwd())

with contextlib.chdir("/Users/janez/Downloads"):
    print(os.getcwd())

print(os.getcwd())

/Users/janez/Desktop/predavanja/p1/drobnarije
/Users/janez/Downloads
/Users/janez/Desktop/predavanja/p1/drobnarije
```

To je uporabno. Če tega ne bi bilo, bi se morali narediti sami. (Eni smo si že: `contextlib.chdir` obstaja šele od Pythona 3.11.)

Da naredimo *upravljalca konteksta* (*context manager*) moramo poznati bodisi razrede bodisi generatorje in dekoratorje. Razrede nekateri že poznate, o generatorjih govorimo v ločenem zapisku. V teh zapiskih pokažimo oba načina. Prvi je poučnejši, drugi prikladnejši.

## Konteksti z razredom

Upravljalca konteksta je razred, ki ima metodi `__enter__` in `__exit__`. `with` pokliče pred vstopom v blok in drugo po izhodu. Prva ima le en argument, `self` (ekvivalent `this`-a v nekaterih drugih jezikih; v Pythonu ga je potrebno eksplicitno navesti med argumenti), druga pa ima tri argumente, ki povedo, ali je bila znotraj bloka sprožena izjema (*exception*) ter kakšna in kje.

```
class NotInVen:
    def __enter__(self):
        print("vstopamo")

    def __exit__(self, exc_type, exc_val, exc_tb):
        print("izstopamo")

print("začetek")

with NotInVen():
    print("v bloku")

print("konec")
```

```
začetek
vstopamo
v bloku
izstopamo
konec
```

To je to. To je že (skoraj) vsa znanost. Ostane le še, da lahko `__enter__` vrne kako vrednost; v `with` jo z `as` priredimo imenu. Takole.

```
class NotInVen:
    def __enter__(self):
        print("vstopamo")
        return 42

    def __exit__(self, exc_type, exc_val, exc_tb):
        print("izstopamo")

print("začetek")

with NotInVen() as x:
    print("v bloku")
    print("x =", x)
```

```
print("konec")
```

```
začetek
vstopamo
v bloku
x = 42
izstopamo
konec
```

### Začasna menjava direktorija

Upravljaec konteksta, ki zamenja trenutni delovni direktorij znotraj bloka, je trivialna zadeva. Če znamo napisati razred v Pythonu, seve.

```
import os

class MojChDir:
    def __init__(self, dir):
        self.dir = dir
        self.saved = None

    def __enter__(self):
        self.saved = os.getcwd()
        os.chdir(self.dir)
```

```

        def __exit__(self, exc_type, exc_val, exc_tb):
            os.chdir(self.saved)

print(os.getcwd())

with MojChDir("/Users/janez/Downloads"):
    print(os.getcwd())

print(os.getcwd())

/Users/janez/Desktop/predavanja/p1/drobnarije
/Users/janez/Downloads
/Users/janez/Desktop/predavanja/p1/drobnarije

```

### Merjenje časa

Za še en primer napišimo kontekst, ki pove, koliko časa se izvaja blok.

```

import time

class Timer:
    def __enter__(self):
        self.start = time.time()

    def __exit__(self, exc_type, exc_val, exc_tb):
        print("Elapsed time: ", time.time() - self.start)

```

Pa pomerimo, če Python prav spi.

```

with Timer():
    time.sleep(1.5)

Elapsed time:  1.505854845046997

```

Kar v redu.

### Začasno spreminjanje funkcij

```

class SuppressPrint:
    def __enter__(self):
        global print
        self.old_print = print

    def print(*args, **kwargs):
        pass

    def __exit__(self, *_):
        global print
        print = self.old_print

```

```
print("Pišem")
```

```
with SupressPrint():  
    print("Ne pišem.")
```

```
print("Spet pišem.")
```

Pišem

Spet pišem.

To ne bo delovalo vedno; če ta razred uvozite iz modula, ne bo imel učinka, ker global ne deluje čisto tako, kot si (najbrž) predstavljate, da deluje.

Kaj pa tole?

```
class TimePrint:  
    def __enter__(self):  
        global print  
        self.old_print = print  
  
        def print(*args, **kwargs):  
            self.old_print(f"{time.time()}: ", *args, **kwargs)  
  
    def __exit__(self, *_):  
        global print  
        print = self.old_print  
  
import time  
import random  
  
with TimePrint():  
    print("Začetek")  
    x = 1 + random.random()  
    time.sleep(x)  
    print("in čez", round(x, 2), "sekund")
```

```
print("Zdaj sem pa spet normalen.")
```

1734281983.302903: Začetek

1734281984.607671: in čez 1.3 sekund

Zdaj sem pa spet normalen.

### Kako se zapirajo datoteke?

Čisto preprosto. Datoteke imajo poleg vseh `read-ov` in `readline-ov` in `write-ov` še metodi `__enter__` in `__exit__`, ki delata približno (ali celo točno) tole:

```
def __enter__(self):  
    return self
```

```
def __exit__(self, exc_type, exc_val, exc_tb):
    self.close()
```

Elegantno, ni?

## Konteksti z generatorjem

Da nam ne bi bilo potrebno pisati razredov, lahko kontekste pišemo tudi s funkcijami. Točneje, generatorskimi funkcijami, ki morajo vsebovati natančno en `yield`. Kar je pred `yield`-om se zgodi pred vstopom v blok, kar po njem, ob izhodu iz bloka. Vrednost, ki jo generiramo, pa je rezultat metode `__enter__`. Tako zapisan kontekst je potrebno *dekorirati* s `contextlib.contextmanager`, tako da pred definicijo funkcije dodamo vrstico `@contextlib.contextmanager`.

Napišimo, recimo, začasno spremembo direktorija.

```
import contextlib

@contextlib.contextmanager
def moj_chdir(dir):
    old_dir = os.getcwd()
    os.chdir(dir)
    yield
    os.chdir(old_dir)

print(os.getcwd())

with moj_chdir("/Users/janez/Downloads"):
    print(os.getcwd())

print(os.getcwd())

/Users/janez/Desktop/predavanja/p1/drobnarije
/Users/janez/Downloads
/Users/janez/Desktop/predavanja/p1/drobnarije
```

Tako je še veliko preprosteje, ni?

Če delamo tako reč čisto zares, je prav, da poskrbimo še za izjeme (*exception*). Detajle preberite v dokumentaciji.

## Črna magija

Pythonovi konteksti vedno uporabljajo razrede. `contextlib.contextmanager` je samo funkcija, ki sama sestavi razred namesto nas. Ljubitelji črne magije naj raziščejo, zakaj tale funkcija dela isto kot `contextlib.contextmanager`.

```
def moj_contextmanager(f):
    class cm:
```

```

    def __init__(self, *args, **kwargs):
        self.g = f(*args, **kwargs)

    def __enter__(self):
        return next(self.g)

    def __exit__(self, *_):
        try:
            next(self.g)
        except StopIteration:
            pass

    return cm

import contextlib

@contextmanager
def moj_chdir(dir):
    old_dir = os.getcwd()
    os.chdir(dir)
    yield
    os.chdir(old_dir)

print(os.getcwd())

with moj_chdir("/Users/janez/Downloads"):
    print(os.getcwd())

print(os.getcwd())

/Users/janez/Desktop/predavanja/p1/drobnarije
/Users/janez/Downloads
/Users/janez/Desktop/predavanja/p1/drobnarije

```