

## Funkcije, ki sestavljajo in vračajo funkcije

Napišimo funkcije, ki pretvarjajo Fahrenheite v Celzije, Celzije v Fahrenheite in, hm kilometre v milje in nazaj.

```
def fahr_v_celz(f):  
    return (f - 32) * 5 / 9
```

```
def celz_v_fahr(c):  
    return c * 9 / 5 + 32
```

```
def km_v_milje(km):  
    return km / 1.60934
```

```
def milje_v_km(m):  
    return m * 1.60934
```

```
fahr_v_celz(45)
```

```
7.222222222222222
```

Rad bi, da vse te funkcije vračajo le dve decimalki. Pravzaprav, ne, rad, bi funkcije `fahr_v_celz2`, `celz_v_fahr2`, `km_v_milje2` in `milje_v_km2`, ki bi vračale taiste reči, vendar le na dve decimalki natančno.

Tega se bomo lotili malo drugače, ne - kot bi pričakovali - z

```
def fahr_v_celz2(x):  
    y = fahr_v_celz(x)  
    return round(y, 2)
```

Pazite tole!

```
def rounder(f):  
    def rounded(x):  
        y = f(x)  
        return round(y, 2)  
  
    return rounded
```

Preden povemo, kako deluje, pogledjmo, kako uporabljati.

```
fahr_v_celz2 = rounder(fahr_v_celz)  
celz_v_fahr2 = rounder(celz_v_fahr)  
km_v_milje2 = rounder(km_v_milje)  
milje_v_km2 = rounder(milje_v_km)  
  
fahr_v_celz2(45)
```

```
7.22
```

Kaj se dogaja? Funkciji `rounder` kot argument podamo funkcijo, recimo `fahr_v_celz`. Znotraj `rounder` je `f` torej isto kot `fahr_v_celz`.

Zdaj pa pogledajmo `rounder`: ta definira funkcijo `rounded`. Če si mislimo, da namesto `f` piše `fahr_v_celz` ... je ta funkcija pravzaprav popolnoma enaka funkciji `fahr_v_celz2`, ki smo jo napisali par celic višje. Funkcija `rounded` torej pravzaprav definira funkcijo `fahr_v_celz2` - ki jo sicer imenuje `rounded`.

In na koncu ... `return rounded`.

Funkcija `rounder` torej ne naredi drugega, kot da definira in vrne funkcijo `fahr_v_celz2`, čeprav pod drugim imenom. Ker jo pokličemo z

```
fahr_v_celz2 = rounder(fahr_v_celz)
```

ima rezultat tega klica `rounder`-ja ime `fahr_v_celz2`. Kot mora.

Klicu `rounder(fahr_v_celz)`, ki torej vrne (v bistvu) `fahr_v_celz2`, sledijo klici `rounder(celz_v_fahr)`, `rounder(km_v_milje)` in `rounder(milje_v_km)`, ki definirajo in vrnejo še ostale tri funkcije.

## Dekoratorji

Funkcija `rounder` nemara ni višek uporabnosti, koncept sam pa. Pogosto si pripravimo (ali pa nam kdo drug pripravi) funkcijo, namenjeno takšnemu "ovijanju" drugih funkcij. Uporabimo jih kar tako, kot da bi zgoraj pisali

```
def fahr_v_celz(f):
    return (f - 32) * 5 / 9

fahr_v_celz = rounder(fahr_v_celz)

def celz_v_fahr(c):
    return c * 9 / 5 + 32

celz_v_fahr = rounder(celz_v_fahr)
```

*# in tako naprej*

Vsako funkcijo torej ovijemo kar takoj po tem, ko jo sestavimo. Ker damo oviti funkciji tudi enako ime, je izvirna funkcija pravzaprav izgubljena, dostopamo lahko le do ovite različice, tiste, ki že zaokroža.

Za to obstaja skrajšana sintaksa.

```
@rounder
def fahr_v_celz(f):
    return (f - 32) * 5 / 9

@rounder
```

```
def celz_v_fahr(c):
    return c * 9 / 5 + 32
```

Stvari, ki smo jo napisali pred definicijo funkcije, rečemo dekorator. Pythonu s tem povemo, naj sicer definira funkcijo, kot je pisano, vendar pravkar definirano funkcijo posleduje dekoratorju (`rounder`); končna definicija funkcije bo tisto, kar vrne `rounder`.

### Primer: memoizacija

Klasičen primer za dekoratorje je memoizacija. Napisali bomo dekorator, ki pokliče ovito funkcijo in vrne njen rezultat. Mimogrede pa si ga še zapomni za naslednjič: če bo še kdo poklical funkcijo z enakim argumentom, bo dekorator vrnil kar shranjeni rezultat, namesto da bi ponovno klical funkcijo.

```
def cache(f):
    stored = {}

    def cached(x):
        if x not in stored:
            stored[x] = f(x)
        return stored[x]

    return cached

@cache
def fahr_v_celz(f):
    print(f"Računam pretvorbo iz {f}")
    return (f - 32) * 5 / 9
```

```
fahr_v_celz(50)
```

```
Računam pretvorbo iz 50
```

```
10.0
```

```
fahr_v_celz(59)
```

```
Računam pretvorbo iz 59
```

```
15.0
```

```
fahr_v_celz(50)
```

```
10.0
```

V zadnjem klicu se ni izpisalo nič, saj je imel dekorator rezultat funkcije za argument 50 že shranjen.

Pretvarjanje Fahrenheitov v Celzije je preprosto, tega nima smisla shranjevati. Kaj pa Fibonaccijeva števila po rekurzivni formuli?

```

klicev = 0

def fibo(n):
    global klicev
    klicev += 1

    if n == 0 or n == 1:
        return 1
    return fibo(n - 1) + fibo(n - 2)

fibo(30)

```

1346269

Funkcija je videti nedolžna, vendar: poglejmo, kolikokrat se je poklicala!

```

klicev

2692537

```

Klicev je dvakrat toliko, kolikor je veliko Fibonaccijevo število. To je, če razmislimo, kar logično. Vendar tu ne bomo razmišljali o tem. Raje bomo razmišljali o tem, kako imenitna stvar je memoizacija.

```

klicev = 0

@cache
def fibo(n):
    global klicev
    klicev += 1

    if n == 0 or n == 1:
        return 1
    return fibo(n - 1) + fibo(n - 2)

```

```

fibo(30)

```

1346269

```

klicev

31

```

To pa je tako logično, da pravzaprav ni o čem razmišljati: izračunati je potrebno vseh 31 Fibonaccijevih števil od 0-tega do 30-tega, vendar nobenega ne računamo več kot enkrat, zato bo klicev točno 31.

Izračunamo lahko tudi stoto Fibonaccijevo število.

```

fibo(100)

573147844013817084101

```

Brez memoizacije to ne bi bilo možno saj bi zahtevalo  $10^{21}$  klicev.

```
26.7 * 10 ** 9 * 365.25 * 24 * 3600 * 1000
```

```
8.4258792e+20
```

Približno en klic za vsako milisekundo obstoja vesolja.

### Posplošena memoizacija

Imenitnost naše funkcije kvari le to, da zna ovijati le funkcije z enim argumentom. Če hočemo oviti takšne s poljubnim številom, je potrebno dekorator malenkost spremeniti.

```
def cache(f):
    stored = {}

    def cached(*x):
        if x not in stored:
            stored[x] = f(*x)
        return stored[x]

    return cached
```

Dodali smo dve zvezdici pred `x`, da se ta ne nanaša več na en sam argument temveč na terko z vsemi argumenti.

Če tega ne razumete, nič hudega. Tako ali tako funkcija že obstaja, najdemo jo v modulu `functools`.

```
from functools import cache
```

### Zaključek

Dekoratorji so uporabni še za marsikaj. Četudi se vam morda zdijo eksotika, jih boste - če boste imeli odprte oči - pogosto zagledali in pogosto tudi pisali.