به نام خدا

زمان بندی یک تابع پایتون

در یکی از مقاله های پیشین (اندازه گیری زمان در پایتون – time.time() در مقابل time.clock())، یاد گرفتیم که چطور از ماژول timeit برای محک زدن یک برنامه استفاده کنیم. از آنجایی که برنامه ای که در اون مقاله زمان بندی کردیم فقط حاوی جمله های خام به جای توابع بود، میخواهیم چگونگی زمان بندی یک تابع را در پایتون بیاموزیم.

زمان بندی یک تابع پایتون بدون آرگومان

ماژول تابع timeit.timeit(stmt, setup, timer, number) چهار آرگومان را می پذیرد:

- stmt که جمله ای است که میخواهید اندازه گیری کنید; مقدار پیش فرض آن 'pass' است.
- Setup که کدی است که قبل از stmt اجرا می شود; مقدار پیش فرض آن 'pass' است.
- Timer که شي timeit.Timer object است; این همیشه یک ارزش مناسب دارد پس نمی خواهد نگران آن باشید.
- Number که تعداد دفعات اجرای stmt است.

جایی که تابع timeit.timeit() تعداد ثانیه هایی که برای اجرای کد طول کشیده را بر می گرداند.

حال فرض کنید می خواهید تابعی به نام costly_func را اندازه بگیرید:

```
def costly_func()
return map(lamdba x: x^2, range(10))
```

می توانید زمان اجرا را با تابع timeit اندازه گیری کنید:

```
import timeit
def costly_func():
        return map(lambda x: x^2, range(10))

# Measute it since costly_func is a callable without argument
timeit.timeit(costly_func)
2.547558069229126
# Measure it using raw statements
timeit.timeit('map(lambda x: x^2, range(10))
```

توجه کنید که ما از دوراه برای اندازه گیری این تابع استفاده کردیم. در روش اول تابع قابل صدا زدن costly_func را رد کردیم ولی در روش دوم جمله های خام تابع costly_func را رد کردیم. اگرچه نوشتن روش اول کمی بیشتر وقت می گیرد، ما معمولا روش اول را ترجیح می دهیم از آن جایی که خوانا تر و پایدار نگه داشتن آن آسان تر است.

زمان بندی یک تابع با آرگومان

می توانیم از decorator ها برای اندازه گیری توابع با آرگومان استفاده کنیم. فرض کنید تابع costly_func ما به صورت زیر باشد:

```
def costly_func(lst):
        return map(lambda x: x^2, lst
```

می توانید با یک decorator ای که به صورت زیر تعیین شده آن‌را اندازه گیری کنید:

```
def wrapper(func, *args, **kwargs):
        def wrapped():
                return func(*args, *kwargs)
        return wrapped
```

حال از این decorator برای گنجاندن تابع costly_func با آرگومان، توی یک تابع بدون آرگومان استفاده کردیم تا بتوانیم توی تابع timeit.timeit آن‌را رد کنیم:

```
def wrapper(func, *args, **kwargs):
        def wrapped():
                return func(*args, **kwargs)
        return wrapped

def costly_func(lst):
        return map(lambda x: x^2, lst)

short_list = range(10)
wrapped = wrapper(costly_func, short_list)
timeit.timeit(wrapped, number= 1000)
0.0032510757446289062
long_list = range(1000)
wrapped = wrapper(costly_func, long_list)
timeit.timeit(wrapped, number=1000)
0.14835596084594727
```

زمان بندی تابع از ماژولی دیگر

فرض کنید که تابع costly_func در یک ماژول دیگری به نام mymodule معین شده است، چطور می توانیم‌زمان آن‌را اندازه گیری کنیم وقتی به صورت محلی قابل دسترس نیست؟ خب، می توانید آن‌را در فضای نام (namespace) محلی وارد(import) کنید یا از آرگومان setup استفاده کنید.

```
# mymodule.py
def costly_fun():
```

```
        return map(lambda x: x^2, range(1000))
```

```
timeit.timeit('costly_func()', setup='from mymodule import costly_func',
number=1000)
0.15768003463745117
# OR just import it in the local namespace
from mymodule import costly_func
timeit.timeit(costly_func, number=1000)
0.15358710289001465
```

خلاصه و نکات

در این مقاله، چگونگی اندازه گیری زمان اجرای یک تابع پایتون‌را توسط timeit.timeit آموختیم. معمولا، ما ترجیح می دهیم تابع پایتون‌را به عنوان شي قابل صدا زدنی وارد و داخل timeit.timeit رد کنیم به دلیل اینکه اینطور نگه داری آن آسان تر است. به علاوه، به خاطر داشته باشید که عدد پیش فرض اجرا ها 1000000 است که میتواند زمان اجرای کلی‌را برای بعضی توابع پیچیده افزایش دهد.

مترجم: علی مرادی

تماس با من: adeadmarshal@gmail.com

لینک منبع