Modul 4 PBF

March 2, 2023

1 Modul 4 Praktikum Pemrograman Berbasis Fungsi

1.1 Sains Data ITERA

1.2 Python Scope

Variabel yang hanya di wilayah fungsi tersebut disebut scope.

Dalam pemrograman, ruang lingkup nama menentukan area program di mana Anda dapat mengakses nama itu dengan jelas, seperti variabel, fungsi, objek, dan sebagainya. Nama hanya akan terlihat dan dapat diakses oleh kode dalam cakupannya. Beberapa bahasa pemrograman memanfaatkan ruang lingkup (Scope) untuk menghindari tabrakan nama dan perilaku yang tidak dapat diprediksi. Paling umum, Anda akan membedakan dua cakupan umum:

1. Lingkup global: Nama yang Anda tetapkan dalam lingkup ini tersedia untuk semua kode Anda.
2. Lingkup lokal: Nama yang Anda tetapkan dalam lingkup ini hanya tersedia atau dapat dilihat oleh kode dalam lingkup.

Scope muncul karena bahasa pemrograman awal hanya memiliki nama global. Dengan nama seperti ini, bagian mana pun dari program dapat mengubah variabel apa pun kapan saja. Untuk bekerja dengan nama global, Anda harus mengingat semua kode pada saat yang sama untuk mengetahui nilai dari nama yang diberikan kapan saja. yang membuat kode pemrograman menjadi lebih sulit di definisikan jika tidak memiliki scope. Kita dituntut untuk menentukan mana variabel didalam scope dan diluar scope.

1.2.1 Local Scope

Variabel yang dibuat didalam fungsi dan milik dari local scope dari fungsi tersebut, dan hanya dapat digunakan didalam fungsi tersebut.

[8]:
```
def myfunc():
  x=1500
  print(x)

myfunc()
```

```
1500
```

1.2.2 Function Inside Function

[9]:
```
def myfunc():
  x = 1500
  def myinnerfunc():
    print(x)
  myinnerfunc()

myfunc()
```

```
1500
```

Fungsi didalam fungsi juga disebut nested function merupakan bagian dari python scope yang memanggil wilayah local dari dalam fungsi itu sendiri.

1.2.3 Global Scope

Variabel yang dibuat dalam main body dari python code adalah global variabel yang berada di wilayah global.

[10]:
```
x = 1500

def myfunc():
  print(x)

myfunc()

print(x)
```

```
1500
1500
```

Fungsi yang mengeluarkan output local x, dan kemudian output global x. Apa perbedaannya?

[11]:
```
x = 300

def myfunc():
  x = 200
  print(x)

myfunc()

print(x)
```

```
200
300
```

1.3 Python Closure

Sebelum masuk pada pengertian closure kita bisa mereview kembali bagaimana proses fungsi dalam fungsi yang diterapkan pada fungsi sebagai objek. Python closure merupakan fungsi yang dapat

memanggil variabel dalam fungsi bahkan ketika fungsi tersebut sudah ditutup (return). Sifat istimewa ini banyak diterapkan pada pemrograman berbasis fungsi python.

Kapan kita menggunakan closures?

Fungsi ini digunakan ketika fungsi yang saling bertaut mereferensikan nilai dalam cakupan terlampir. kriteria yang harus dipenuhi ketika membuat closure adalah sebagai berikut: 1. Kita harus memiliki nested function (atau fungsi dalam fungsi) 2. nested function harus merujuk ke nilai yang didefinisikan didalam fungsi yang sudah ditutup 3. penutupan dari fungsi yang dibangun harus kembali ke nested function

Closure dapat keluar dari nilai global dan menyediakan beberapa bentuk penyembunyian data daripada menggunakan objek oriented. saat menggunakan beberapa metode atau satu metode saja pada sebuah class, biasanya closure dapat digunakan sebagai solusi alternatif. Tetapi ketika jumlah atribut dan metode semakin membesar, tidak disarankan menggunakan closure, dan lebih baik menggunakan class.

[1]:
```
def greet(name):
    # inner function
    def display_name():
        print("Hi", name)

    # call inner function
    display_name()
# call outer function
greet("Data")
```

```
Hi Data
```

Pada contoh di atas, kita mendefinisikan display_name() function di dalam greet() function. yang mana pada display_name() merupakan nested function. fungsi ini berkerja sama seperti fungsi seperti biasanya. dan akan di eksekusi ketika display_name di panggil di dalam greet() function. Contoh lainnya dapat diterapkan ke lambda function.

[2]:
```
def greet():
    # variable defined outside the inner function
    name = "Sains Data"

    # return a nested anonymous function
    return lambda: "Hi " + name

# call the outer function
message = greet()

# call the inner function
print(message())
```

```
Hi Sains Data
```

Pada contoh di atas di jelaskan bahwa ketika memanggil fungsi diluar `message = greet()` dan akhirnya fungsi kembali ke `massage()`

[3]:
```
def calculate():
    num = 1
    def inner_func():
        nonlocal num
        num += 2
        return num
    return inner_func

# call the outer function
odd = calculate()

# call the inner function
print(odd())
print(odd())
print(odd())

# call the outer function again
odd2 = calculate()
print(odd2())
```

```
3
5
7
3
```

`odd = calculate()` kode ini mengeksekusi fungsi diluar dari function calculate() dan kembali ke closure dengan bilangan ganjil. Hal ini merupakan alasan bahwa kita dapat mengakses variabel num dari fungsi `calculate()` meskipun setelah menyelesaikan outer function. Dan setelah kita kembalikan fungsi tersebut menjadi `odd2 = calculate()` lalu output yang dihasilkan dikembalikan ke urutan yang pertama.

[4]:
```
def make_multiplier_of(n):
    def multiplier(x):
        return x * n
    return multiplier

# Multiplier of 3
times3 = make_multiplier_of(3)

# Multiplier of 5
times5 = make_multiplier_of(5)

# Output: 27
print(times3(9))

# Output: 15
print(times5(3))
```

```
# Output: 30
print(times5(times3(2)))
```

```
27
15
30
```

[7]:
```
def foo(y):
    def bar(x):
        return (x+y)
    return bar

print(foo(3)(2))

#Lambda
foo = lambda y : (lambda x : x + y)

print(foo(2)(3))
```

```
5
5
```