Modul 1 Praktikum Pemrograman Berbasis Fungsi

February 15, 2023

1 Modul Praktikum Pemrograman Berbasis Fungsi

Tujuan Praktikum: 1. Memahami pengenalan bahasa pemrograman Python berbasis fungsi

2. Memahami konsep dasar paradigma pemrograman berbasis fungsi

2 Pemrograman Python Berbasis Fungsi

Fungsi adalah blok kode yang hanya berjalan ketika dipanggil. Anda dapat meneruskan data, yang dikenal sebagai parameter, ke dalam suatu fungsi. Sebuah fungsi dapat mengembalikan data sebagai hasilnya. Fungsi terdiri dari input , process dan output. Input dari sebuah fungsi dalam python disebut argumen/parameter, sedangkan output disebut dengan return value. Fungsi didefinisikan dengan keyword def, dan dapat dipanggil dalam program utama untuk mengeksekusi blok kode yang telah didefinisikan tersebut. Blok kode tersebut adalah function body nya. Function body adalah process yang dilakukan oleh fungsi saat fungsi itu dipanggil.

[9]:
```
def square(x):
  y = x*x
  return y
```

Fungsi tersebut bernama square. fungsi square memiliki input parameter/argumen yaitu variabel x. Proses Assignment y=x*x adalah function body dari square. Output dari fungsi square adalah y, sebagai hasil kuadrat dari input x. Keyword return menandakan ekspresi yang dikeluarkan oleh fungsi tersebut sebagai output.

2.1 Jenis Fungsi dalam python

Dalam python, secara umum terdapat 3 jenis fungsi, yaitu:

1. Built-in function , yaitu fungsi yang sudah ada dalam python sehingga dapat langsung dipanggil.
2. User-Defined function, yaitu fungsi yang kita definisikan sendiri.
3. Function defined in a module, fungsi yang didefinisikan dalam suatu modul python. Untuk memanggil fungsi tersebut kita harus melakukan import statement terhadap modul tersebut terlebih dahulu

[10]:
```
# Built in function max() -> mengeluarkan nilai paling tinggi dalam suatu list
L = [1,2,3,8,5,98,1,0,-2]
```

```
max(L)
```

[10]: 98

Contoh user-defined by function:

[11]:
```
def cube(x):
  return x*x*x
cube(10)
```

[11]: 1000

Contoh function defined in a module:

[12]:
```
import math # modul yang di import adalah math
math.sqrt(100) # fungsi sqrt dalam modul math
```

[12]: 10.0

2.2 Fungsi sebagai cara untuk modularisasi Algoritma

Fungsi dapat digunakan untuk menjadikan satu blok kode, menjadi beberapa part. Proses break-down tersebut menjadikan kode python kita menjadi lebih modular.

[13]:
```
# Code python untuk menghitung faktor dari sebuah bilangan n
n = 64
faktors = []
for i in range(1,n+1):
  if n % i == 0:
    faktors.append(i)
print(faktors)
```

```
[1, 2, 4, 8, 16, 32, 64]
```

[14]:
```
# Modularisasi Code nya
def habis_dibagi(n,i):
  return n % i == 0
def faktor(n):
  temp = []
  for i in range(1,n+1):
    if habis_dibagi(n,i):
      temp.append(i)
  return temp
faktor(64)
```

[14]: [1, 2, 4, 8, 16, 32, 64]

[15]:
```
faktor(25)
```

[15]: [1, 5, 25]

2.3 Anonymous Function

Fungsi dalam python dapat bersifat anonymous. Anonymous function adalah user defined function yang didefinisikan tanpa menggunakan nama fungsi. Fungsi Anonymous dalam python didefinisikan menggunakan keyword lamba dan umumnya dikenal dengan lambda function.

[17]:
```
# Normal User Defined Function
def rata(L):
  res = 0
  for el in L:
    res = res + el
  res = res/ len(L)
  return res
L = [1,2,3,4,5,6,7,8,9,10]
rata(L)
```

[17]: 5.5

[18]:
```
# Lambda Function
rata_lambda = lambda L : sum(L) / len(L)
rata_lambda(L)
```

[18]: 5.5

Lambda function dapat menerima lebih dari 0 hingga n argument, akan tetapi hanya boleh berisi satu ekspresi saja seperti fungsi diatas. Blok body fungsi dari fungsi rata harus disederhanakan menjadi satu ekspresi terlebih dahulu agar bisa diubah menjadi fungsi lambda. Contoh lambda function tanpa argumen:

[20]:
```
greet = lambda : 'Hello World'
greet()
```

[20]: 'Hello World'

Contoh lambda function lebih dari 1 argument:

[21]:
```
greet = lambda name,gender: 'Hello ' + ('Mr.' if gender=='M' else 'Ms.') + name
greet('Satria','M')
```

[21]: 'Hello Mr.Satria'

Dalam python, karena fungsi dianggap sebagai first-class object. Maka fungsi dapat berada di satu ekspresi sehingga jika ada proses rumit dan panjang dalam body function yang sulit diubah menjadi satu ekspresi, kita dapat terlebih dahulu membuat fungsi bantuan yang nanti akan menjadi ekspresi dari lambda function. Contoh:

[19]:
```
def helper(L):
  res = 0
  for el in L:
    res = res + el
```

```
  return res
rata = lambda L: helper(L) / len(L)
rata(L)
```

[19]: 5.5

2.4 Ternary Expression and Operator

Operator ternary juga dikenal sebagai ekspresi kondisional adalah operator yang mengevaluasi sesuatu berdasarkan kondisi yang benar atau salah. Itu ditambahkan ke Python di versi 2.5. Ini hanya memungkinkan pengujian suatu kondisi dalam satu baris menggantikan multiline if-else membuat kode menjadi compact. Operator ternary dapat membantu jika ada conditional statement dalam sebuah fungsi lambda. Contoh:

[22]:
```
# mencari minimum dari 2 bilangan
a,b = 10,20
if a>b:
  min = b
else:
  min = a
min
```

[22]: 10

[23]:
```
# mencari minimum dari 2 bilangan menggunakan operator ternary
a, b = 10, 20
minim = a if a < b else b
minim
```

[23]: 10

Ternary operator juga dapat digunakan seperti nested if-else untuk banyak kondisi. Contoh:

[29]:
```
# menggunakan nested if-else
a, b = 10, 20
if a != b:
  if a > b:
      print("a lebih besar dari b")
  else:
      print("b lebih besar dari a")
else:
  print("a dan b bernilai sama")
```

```
b lebih besar dari a
```

[30]:
```
# nested ternary operator
a, b = 10, 20
print ("a dan b bernilai sama" if a == b else "a lebih besar dari b"
if a > b else "b lebih besar dari a")
```

```
b lebih besar dari a
```

Contoh Penggunaan ternary operator dalam lambda function:

[31]:
```
# Mengecek bilangan genap atau ganjil
f = lambda x: 'genap' if x%2==0 else 'ganjil'
f(10)
```

[31]:
```
'genap'
```

2.5 Lambda function direct calling

Lambda function dapat secara langsung dipanggil dan tidak harus di assign kepada sebuah variabel. Contoh:

[32]:
```
(lambda x:'genap' if x%2==0 else 'ganjil')(10)
```

[32]:
```
'genap'
```

Contoh Functool

[ ]:
```
val = [1, 2, 3, 4, 5, 6]

# Multiply every item by two
list(map(lambda x: x * 2, val)) # [2, 4, 6, 8, 10, 12]
# Take the factorial by multiplying the value so far to the next item
reduce(lambda: x, y: x * y, val, 1) # 1 * 1 * 2 * 3 * 4 * 5 * 6
```

3 Pemrograman Berbasis Fungsi pada Analisis Data

3.1 Karakteristik Pemrograman Berbasis Fungsi

- Bahasa pemrograman fungsional dirancang berdasarkan konsep fungsi matematika yang menggunakan ekspresi bersyarat dan rekursi untuk melakukan perhitungan.
- Pemrograman fungsional mendukung fungsi tingkat tinggi (Higher order Function) dan fitur Lazy Evaluation.
- Bahasa pemrograman fungsional tidak mendukung kontrol aliran (Flow Control) seperti pernyataan loop dan pernyataan bersyarat seperti Pernyataan If-Else dan Switch. pemrograman ini langsung menggunakan fungsi dan panggilan fungsional.
- Seperti OOP, bahasa pemrograman fungsional mendukung konsep populer seperti Abstraksi, Enkapsulasi, Pewarisan, dan Polimorfisme (Abstraction, Encapsulation, Inheritance, and Polymorphism)

3.1.1 Data Preprocessing dengan pemrograman preosedural

[9]:
```
import pandas as pd
import numpy as np
np.random.seed(42)
```

```
def create_random_dataframe(nb_rows=1000, percent_na_per_column=[0.01, 0.1, 0.
 ↪9]):
    df = pd.DataFrame()
    for i, percent_na in enumerate(percent_na_per_column):
        values = [np.random.random() if np.random.random() > percent_na else np.
 ↪nan for _ in range(nb_rows)]
        df[f'val_{i}'] = values
    df.index = pd.date_range('2023/02/14', periods=nb_rows, freq='H')
    return df


def preprocess(df, na_percentage=0.2, resampling_str='2H'):
    # remove columns with too many nas
    na_df = df.isna().sum() / len(df)
    list_col_to_keep = na_df[na_df < na_percentage].index
    df = df[list_col_to_keep]

    # resampling
    df = df.resample(resampling_str).mean()

    # filling missing values
    df = df.interpolate().ffill().bfill()
    return df

random_df = create_random_dataframe()
res = preprocess(random_df)
res
```

3.1.2 Data Preprocessing dengan pemrograman fungsi

[38]:
```
import pandas as pd
import numpy as np
np.random.seed(42)

def create_random_dataframe(nb_rows=1000,percent_na_per_column=[0.01, 0.1, 0.
 ↪9]):
    df = pd.DataFrame()
    for i, percent_na in enumerate(percent_na_per_column):
        values = [np.random.random() if np.random.random()> percent_na  else np.
 ↪nan for _ in range(nb_rows)  ]
        df[f'val_{i}'] = values
    df.index = pd.date_range('2023/02/14', periods=nb_rows, freq='H')
    return df

def create_col_with_too_many_nas_remover(na_percentage=0.2):
    def remover(df):
        na_df = df.isna().sum()/len(df)
```

```
        list_col_to_keep = na_df[na_df<na_percentage].index
        return df[list_col_to_keep]
    return remover

def create_resampler(resampling_str):
    def resampler(df):
        return df.resample(resampling_str).mean()
    return resampler

def fill_na(df):
    return df.interpolate().ffill().bfill()


random_df = create_random_dataframe()
remover = create_col_with_too_many_nas_remover()
resampler = create_resampler('2H')
res = fill_na(resampler(remover(random_df)))
res
```

Tugas individu: Analisis skrip code di atas sebelum dan sesudah di transformasi menjadi functional programming. Test dengan time init

3.1.3 Bagaimana Mengubah Prosedural Programming ke Functional Programming

[ ]:
```
# procedural code
starting_number = 96

# get the square of the number
square = starting_number ** 2

# increment the number by 1
increment = square + 1

# cube of the number
cube = increment ** 3

# decrease the cube by 1
decrement = cube - 1

# get the final result
result = print(decrement)
```

[ ]:
```
# define a function `call` where you provide the function and the arguments
def call(x, f):
    return f(x)

# define a function that returns the square
```

```
square = lambda x : x*x

# define a function that returns the increment
increment = lambda x : x+1

# define a function that returns the cube
cube = lambda x : x*x*x

# define a function that returns the decrement
decrement = lambda x : x-1

# put all the functions in a list in the order that you want to execute them
funcs = [square, increment, cube, decrement]

# bring it all together. Below is the non functional part.
# in functional programming you separate the functional and the non functional␣
 ↪parts.
from functools import reduce # reduce is in the functools library
print(reduce(call, funcs, 96))
```

3.2 Tugas Jurnal Kelompok

*1 Buatlah fungsi komposisi dengan nilai

$$f(x) = 4^x, g(x) = x^x$$

dan

$$h(x) = 3 + x$$

dengan pemrograman prosedural tiga fungsi dan pemrograman fungsional dengan lambda.

*2 Buatlah pemrograman berbasis objek pada soal no.1

*3 Berikan Analisis Komparasi soal 1 dan 2

*4 Buatlah satu penyelesaian Luas Segitiga dengan menggunakan Pemrograman berorientasi Objek dan Pemrograman berbasis Fungsi. uji dengan membandingkan mana yang tercepat komputasinya.

Special Soal 5: Diberikan Algoritma Metode Bisection berikut: 1. start

2. Define function f(x)
3. Choose initial guesses x0 and x1 such that f(x0)f(x1) < 0
4. Choose pre-specified tolerable error e.
5. Calculate new approximated root as x2 = (x0 + x1)/2
6. Calculate f(x0)f(x2)
    a. if f(x0)f(x2) < 0 then x0 = x0 and x1 = x2
    b. if f(x0)f(x2) > 0 then x0 = x2 and x1 = x1
    c. if f(x0)f(x2) = 0 then goto (8)

7. if |f(x2)| > e then goto (5) otherwise goto (8)
8. Display x2 as root.
9. Stop

Buatlah prosedural programming dari persoalan di atas dan ubahlah dengan functional programming. lakukan dengan batas toleransi

$$10^{-}5$$

. dengan fungsi yang diberikan sebagai berikut:

$$x - 2^x$$

dan nilai a dan b adalah di antara 0 dan 1 (nilai antara tidak boleh 0 dan 1)