Review Algoritma Pemrograman

Indah Agustien Siradjuddin

Review Materi dasar Algoritma Pemrograman

Berikut adalah pengulangan kembali materi-materi dasar (termasuk contoh code) yang terdapat pada mata kuliah Algoritma Pemrograman :

1. Variabel dan Tipe Data
2. Jenis Algoritma
3. String-List-Tuple
4. Dictionaries
5. Function
6. Module

Variabel dan Tipe Data

Variabel adalah sebuah nama atau identitas yang merepresentasikan suatu nilai dengan tipe data tertentu.
Berbagai macam tipe data dasar yang dikenal dalam bahasa pemrograman Python, antara lain:

```
- Integer, merupakan data numerik
- Float, merupakan data numerik yang berbentuk real
- String, merupakan data teks
- Boolean, merupakan data yang hanya memiliki dua nilai saja, yaitu True
dan False
```

Code

Berikut adalah contoh-contoh code inisialisasi dan informasi type data dengna menggunakan syntax **type** untuk masing-masing tipe data

In [ ]:
```
num = 100
type(num)
```

In [ ]:
```
realNum = 0.9999
type(realNum)
```

In [ ]:
```
data="Hello World"
type(data)
```

In [ ]:
```
boolDat=True
type(boolDat)
```

Operasi Data

Setiap variabel dapat dioperasikan dengan menggunakan operasi tertentu, akan tetapi, operasi dapat dilakukan hanya pada variabel-variabel yang memiliki tipe data yang sama atau hampir sama (seperti integer dengan float)

Code

Berikut adalah contoh code untuk operasi antara variabel dengan tipe data yang sama

In [ ]:
```
#Operasi antara variabel bertipe integer
a=10
b=5
hasil=a+b
print('hasil=',hasil)
```

In [ ]:
```
#Operasi antara variabel bertipe float
a=5.5
b=1.2
hasil=a+b
print('hasil=',hasil)
```

In [ ]:
```
#Operasi antara variabel bertipe string
#concatenation
a='struktur'
b='data'
hasil=a+' '+b
print('hasil=',hasil)
```

Code

Berikut adalah contoh code untuk operasi antara variabel dengan tipe data yang tidak sama

In [ ]:
```
#Operasi antara variabel bertipe integer dengn variabel bertipe float
a=10
b=5.5
hasil=a+b
print('hasil=',hasil)
```

In [ ]:
```
#Operasi antara variabel bertipe integer dengn variabel bertipe string
a='struktur data'
b=2
hasil=a*b
print('hasil=',hasil)
```

[Kembali ke Menu Awal]

Algoritma

Terdapat beberapa jenis algoritma dasar, yaitu:

- *Sequential*
- *Branching*
- *Iteration*

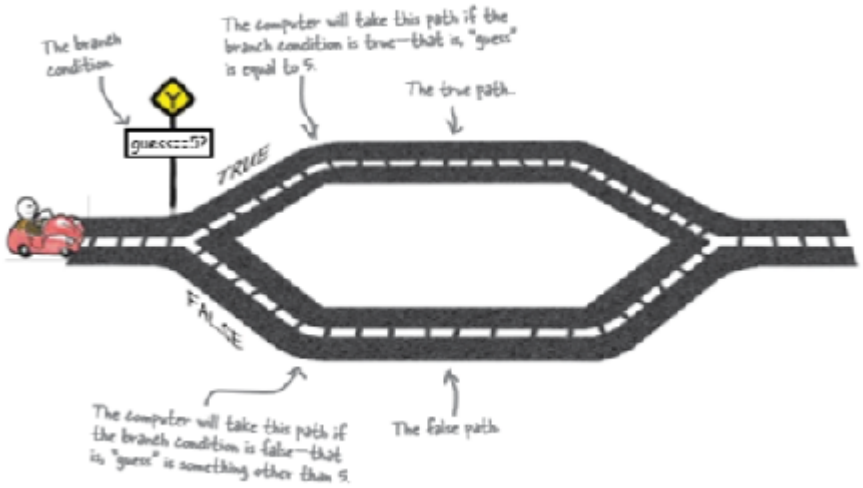
Sequential

Algoritma *sequential* merupakan algoritma yang harus dikerjakan berurutan, mulai dari langkah pertama sampai dengan langkah terakhir.
Ilustrasi algoritma *sequential* ini dapat dilihat pada Gambar 1.

Gambar 1. Algoritma *Sequencial* (Gambar, courtesy:Barry and Griffiths 2009)

Branching / Selection / Pilihan

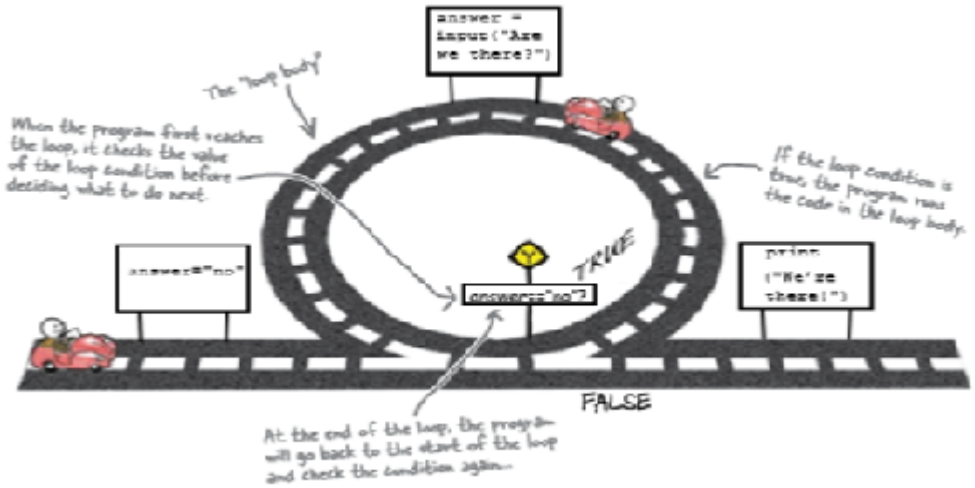
Algoritma *branching* atau *selection* ini merupakan algoritma yang didalamnya terdapat pilihan. Pada algoritma *branching* terdapat kondisi **True** atau **False**. Jika memenuhi kondisi **True** maka syntax pada percabangan **True** yang akan diekseskusi, begitu juga sebaliknya.
Ilustrasi algoritma ini dapat dilihat pada Gambar 2.



Gambar 2. Algoritma *Branches*(Gambar, courtesy: Barry and Griffiths 2009)

Iteration / Looping / Perulangan

Pada Algoritma ini, syntax akan dieksekusi secara berulang-ulang selama kondisi bernilai **True**. Jika kondisi bernilai **False** maka proses iterasi akan berhenti. Ilustrasi Algoritma ini dapat dilihat pada Gambar 3.



Gambar 3. Algoritma *Iteration* (Gambar, courtesy:Barry and Griffiths 2009)

Code

Berikut adalah contoh code untuk masing-masing jenis algoritma

In [ ]:

```
# Algoritma Sequence - Konversi Kurs Mata Dollar ke Rupiah
dollar=int(input('Jumlah Dollar = '))
rupiah=dollar*15000
print(dollar,'$ = Rp.',rupiah)
```

In [ ]:
```
# Algoritma Branching - Penentuan jenis bilangan
num=int(input('Masukkan bilangan = '))
if num%2==0 :
    print(num, ' adalah bilangan genap')
else:
    print(num, ' adalah bilangan ganjil')
```

In [ ]:
```
# Algoritma Branching - Penentuan jenis bilangan
num=int(input('Masukkan bilangan = '))
if num%2==0 :
    print(num, ' adalah bilangan genap')
if num%2!=0:
    print(num, ' adalah bilangan ganjil')
```

In [ ]:
```
# Algoritma Iteration - Menampilkan sejumlah n bilangan genap
num=int(input('bilangan genap Tertinggi = '))
count=1
i=0
while i<=num:
    if i%2==0:
        print(count,'. ',i)
        count=count+1
    i=i+1
```

Latihan - 1

1. Buat code untuk menghitung factorial suatu bilangan
2. Buat code untuk menentukan apakah suatu bilangan adalah bilangan prima

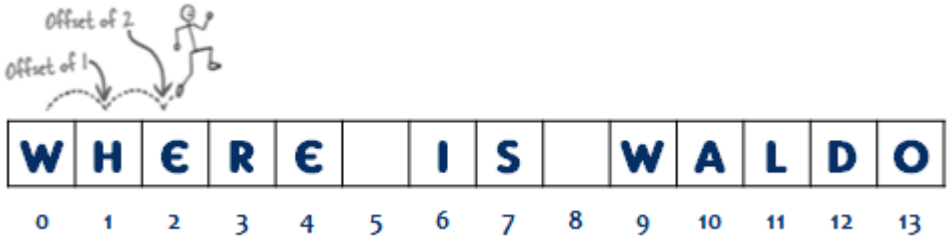
In [ ]:
```
num=int(input('?='))
count=0
for i in range(1,num+1):
    if num%i==0:
        count+=1

if num==1 or count>2:
    print('Bukan bilangan prima')
else:
    print('Bilangan prima')
```

Kembali ke Menu Awal

String, List, dan Tuple

String

String merupakan salah satu tipe data dalam suatu bahasa pemrograman yang terdiri dari beberapa *character*. Pada tipe data string di python, dikenal **offset**, yang menunjukkan *character* ke- dari posisi awal string (yaitu **offset** 0) .
Contoh offset pada suatu variabel dengan nilai 'Where is Waldo' dapat dilihat pada Gambar 3.



Gambar 3. Tipe data String dan offset

Suatu karakter pada string dapat diakses dengan menggunakan syntax namaVariabel[offset], atau jika terdapat beberapa karakter yang harus diakses, dapat menggunakan syntax namaVariabel[offsetAwal:offsetAkhir-1]

Code

Berikut adalah contoh inisialisasi variabel dengan tipe data string, dan cara pengaksesan satu dan lebih dari satu karakter

In [ ]:
```
data="Where is Waldo ?"
print(data[12]) # Satu karakter
temp=data[9:14] # lima karakter
print(temp)
```

Elemen atau anggota pada String bersifat **Immutable** atau tidak dapat dirubah dengan melalui operasi *assignment*. Akan tetapi nilai string dapat dirubah melalui method *replace* yang terdapat pada string

Code

Berikut contoh code untuk proses perubahan nilai melalui operasi assignment, dan melalui method pada string

In [ ]:
```
#immutable, tidak dapat dirubah melalui assignment operator
data="Where is Waldo ?"
data[13]='i'
print(data)
```

In [ ]:
```
#nilai dirubah melalui method 'replace'
data="Where is Waldo ?"
data.replace('o','i')
```

List

List adalah struktur data yang terdiri dari beberapa elemen atau anggota list dengan berbagai tipe data. Masing-masing elemen dipisahkan oleh ','. Seperti $a = [6, 10, 23]$
Didalam list terdapat **index** yang menunjukkan urutan elemen dari suatu list. Index dari list ini dimulai dari '0'. Untuk mengakses elemen pada suatu list, dilakukan dengan cara yang sama dengan tipe data string.

Code

Berikut ini adalah contoh inisialisasi variabel dengan tipe data list, dan cara mengakses satu dan lebih dari satu elemen pada list

In [ ]:
```
arrData=[1,2,'python',0.8,'numpy']
print (arrData)
print(arrData[1])
print(arrData[1:4])
```

Berbeda dengan tipe data string, anggota atau elemen dari list ini bersifat **Mutable**, atau nilai elemen pada list dapat dirubah melalui operasi *assignment*, yaitu menggunakan operator *assignment* '='

Code

Berikut adalah contoh code untuk merubah nilai elemen dari suatu list

In [ ]:
```
arrData=[1,2,'python',0.8,'numpy']
print(arrData)
arrData[2]='Java'
print(arrData)
```

List 2D adalah list yang berbentuk dua dimensi, yaitu bentuk data seperti halnya matriks dua dimensi, yang memiliki baris dan kolom. Misalkan terdapat suatu list dua dimensi, dengan 3 baris, dan dua kolom, maka inisialisasi list tersebut dapat dilakukan dengan cara:
$namaVariabel = [[a11, a12], [a21, a22], [a31, a32]]$
Untuk mengakses data yang terdapat pada list ini, diperlukan dua buah indeks, yaitu indeks yang menyatakan baris, dan indeks yang menyatakan kolom. Indeks dimulai dari 0.

Code

Berikut adalah contoh pembentukan list 2D, dan cara mengakses data yang terdapat pada list tersebut

In [ ]:
```
#list 2D, dengan 2 baris, 3 kolom
arr2=[[1,2,3],[4,5,6]]
print(arr2)
print(arr2[0][2])
```

Tuple

Tuple, sama halnya dengan list, tuple ini terdiri dari beberapa elemen, dan elemen tersebut dapat terdiri dari berbagai tipe. Jika pada list, representasi anggota menggunakan kurung siku atau [a,b,c,...,d], maka pada tuple menggunakan kurung atau (a, b, c, ..., d). Cara pengaksesan elemen atau anggotanya, dilakukan dengan menuliskan index tuple.

Code

Berikut adalah contoh code untuk inisialisasi tuple dan akses elemen pada tuple

In [ ]:
```
tupData=(1,2,'python',0.8,'numpy')
print(tupData)
print(tupData[2])
print(tupData[1:3])
```

Seperti halnya tipe data string, elemen pada tuple bersifat **immutable**, nilai anggota tidak dapat dirubah. Oleh karena itu tuple biasanya digunakan untuk merepresentasikan kumpulan data konstan, dan untuk index *dictionary* (karena index tidak boleh dirubah).

Code

Berikut contoh code untuk menunjukkan sifat immutable pada tuple

In [ ]:
```
tupData=(1,2,'python',0.8,'numpy')
print(tupData)
print(tupData[2])
tupData[2]='Java'
print(tupData)
```

[Kembali ke Menu Awal]

Dictionaries

Dictionaries ini hampir sama dengan struktur data **List** hanya saja, jika pada list menggunakan **index berupa integer** untuk menunjukkan posisi elemen di dalam list, maka pada **dictionaries** ini, maka index pada *dictionaries* yang digunakan tidak hanya bertipe **integer**, akan tetapi **string** juga dapat digunakan sebagai index. Sehingga untuk keperluan tertentu, dictionaries ini akan lebih cocok digunakan untuk merepresentasikan suatu data.

Pada *dictionaries*, index ini disebut juga dengan **keys**. Jika pada list menggunakan '[...]', dan tuple menggunakan (...), maka dictionaries menggunakan kurung kurawal untuk mendefinisikan anggotanya, yaitu {...}. Antara satu elemen dengan elemen yang lain, dipisahkan dengan koma ','. Inisialisasi variabel yang berbentuk *dictionaries* ini dapat dilakukan dengan dua cara, yaitu

1. Menuliskan semua anggotanya secara langsung, namaVariabel={key1:data1, key2:data2,...}
2. Menuliskan satu-persatu anggotanya

Code

Berikut adalah contoh inisialisasi *dictionaries* dengan dua cara tersebut

In [ ]:
```
a=[4,2,3,1,1]
a[3]=7
```

In [ ]:
```
#Cara-1
studData={'001':'Ranti','002':'Diana','003':'Budi','004':'Eri'}
print(studData)
```

In [ ]:
```
#Cara-2
studentData={}
studentData['001']='Fatimah'
studentData['002']='Sofiah'
studentData['003']='Ahmad'
studentData['005']='Ali'
print(studentData)
```

Seperti halnya *list*, *dictionaries* juga dapat berbentuk 2D, yaitu elemen-elemen disusun pada baris dan kolom tertentu, sehingga untuk mengakses suatu data, diperlukan dua buah indeks. Karena dibutuhkan dua buah index pada dictionaries ini, maka index harus berbentuk tuple (karena sifat index yang tidak boleh dirubah nilainya), sehingga index dictionaries 2D ini ditunjukkan dengan pasangan data atau tuple, '(indeks1,indeks2)'

Salah satu implementasi Dictionaries ini adalah **Sparse Matrix**. Sparse Matrix merupaan matrix yang memiliki banyak nilai nol, sehingga jika menggunakan list, akan banyak memori yang dibutuhkan. Sedangkan dictionaries, hanya menyimpan yang dibutuhkan saja. Misalkan terdapat sebuah matriks dengan ukuran 4 x 4, yang berarti terdapat 16 elemen pada matriks tersebut. Hanya saja semua elemen bernilai 'nol' dan hanya indeks tertentu yang bernilai bukan 'nol', seperti :

$$\mathbf{Mat} = \begin{vmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 3 \end{vmatrix}$$

Jika data matriks tersebut direpresentasikan oleh list, maka syntax pembentukan list, dapat dilakukan sebagai berikut:
$Mat = [[0,0,0,1],[0,0,0,0],[0,2,0,0],[0,0,0,3]]$

Akan tetapi, jika menggunakan dictionaries (dimana index tidak harus berurutan), maka matriks tersebut dapat disimpan dengan syntax berikut:
$Mat = \{(0,3):1,(2,1):2,(3,3):3\}$

Oleh karena itu, dengan representasi dictionaries ini lebih efisien untuk kasus tertentu

Code

Berikut contoh code pembuatan *sparse matrix * dengan menggunakan dictionary

In [ ]:
```
Mat = {(0,3): 1, (2, 1): 2, (3, 3): 3}
print(Mat)
Mat[0,2]=4 #penambahan data baru
print(Mat)
#pengecekan data pada index (ind1,ind2), jika tidak terdapat data, maka retur
#jika terdapat data maka return value=adalah data
cek=Mat.get((0,1))
print(cek)
cek=Mat.get((2,1))
print('(2,1)=',cek)
cek=Mat.get((1,3))
print('(1,3)=',cek)
```

[Kembali ke Menu Awal]

Function

Semakin rumit permasalahan yang harus dipecahkan dengan menggunakan bahasa pemrograman maka semakin kompleks program yang dibuat. Secara umum, semakin kompleks algoritma, maka semakin sulit untuk dibaca dan diperbaiki jika terdapat kesalahan atau penambahan *syntax* baru
Oleh karena itu dibutuhkanlah sebuah **function**, dengan function maka syntax-syntax pada program dikelompokkan berdasarkan fungsinya masing-masing. Dengan pengelompokan ini, maka program akan lebih mudah untuk dibaca maupun diperbaiki.
Selain itu, dengan pembentukan **function** ini maka, jika dibutuhkan eksekusi perinta-perintah yang sama, maka dapat dilakukan hanya dengan cara memanggil *function* tersebut, tanpa perlu menulis kembali syntax-syntax yang sama.

Terdapat dua hal penting yang harus diperhatikan, yaitu

1. Parameter/argumen : merupakan nilai yang dikirim oleh syntax pemanggil *function*
2. Return Value : merupakan nilai yang dihasilkan oleh *function*, dan dikirim kembali ke pemanggil *function*

Secara umum sebuah function dapat didefinisikan sebagai berikut:

```
def NamaFungsi(Argumen,...):
    subcode
    subcode
    return returnValue
```

Untuk memanggil fungsi, dilakukan dengan menuliskan nama fungsi

Code

Berikut contoh sederhana *function* tanpa parameter, *function* dengan parameter, dan *function* dengan *return value*

In [ ]:

```
# Function tanpa parameter

def addNumbers():
    a=int(input('Bilangan pertama = '))
    b=int(input('Bilangan kedua = '))
    print('Hasil = ',a+b)

#Main program
addNumbers()#memanggil fungsi addNumbers agar dieksekusi
```

In [ ]:

```
# Function dengan parameter

def addNumbers(a,b):
    #print('Hasil = ',a+b)
    hasil=a+b
#Main program
num1=int(input('Bilangan pertama = '))
num2=int(input('Bilangan kedua = '))
hasil=addNumbers(num1,num2)#memanggil fungsi addNumbers agar dieksekusi
print(hasil)
```

In [ ]:

```
# Function dengan parameter dan return value
def addNumbers(a,b):
    hasil=a+b
    return hasil

def cekGenap(num):
    if num%2==0:
        return True
    else:
        return False

#Main program
num1=int(input('Bilangan pertama = '))
num2=int(input('Bilangan kedua = '))
result=addNumbers(num1,num2)#memanggil fungsi addNumbers agar dieksekusi
print('Hasil Penjumlahan= ', result)
if cekGenap(result):
    print(result,' adalah Bilangan Genap')
else:
    print(result,' adalah Bilangan Ganjil')
```

In [ ]:
```
def addMat1D(a,b):
    if len(a)==len(b):
        hasil=[]
        for i in range(len(a)):
            hasil.append(a[i]+b[i])
        return hasil
    else:
        return 'Ukuran Tidak Sama'
def createMat1D(size):
    hasil=[]
    for i in range(size):
        temp='Data ke-'+str(i)+'='
        hasil.append(int(input(temp)))
    return hasil
```

In [ ]:
```
a=createMat1D(3)
b=[1,2,3]
c=addMat1D(a,b)
print(a,b,c)
```

Latihan - 2

Buat program untuk menghitung perkalian dua buah matriks yang berukuran 3 x 3 seperti berikut :

$$\mathbf{A}_1 \times \mathbf{A}_2 = \begin{vmatrix} \mathbf{a1} & \mathbf{b1} & \mathbf{c1} \\ \mathbf{d1} & \mathbf{e1} & \mathbf{f1} \\ \mathbf{g1} & \mathbf{h1} & \mathbf{i1} \end{vmatrix} \times \begin{vmatrix} \mathbf{a2} & \mathbf{b2} & \mathbf{c2} \\ \mathbf{d2} & \mathbf{e2} & \mathbf{f2} \\ \mathbf{g2} & \mathbf{h2} & \mathbf{i2} \end{vmatrix}$$

Buatlah fungsi-fungsi untuk membuat matriks, menghitung perkalian dua buah matriks

In [ ]:
```
def createMat2D(bar,kol):
    hasil=[[] for i in range(bar)]
    for i in range(bar):
        for j in range(kol):
            tempStr='Data ke-'+str(i)+','+str(j)+'='
            hasil[i].append(int(input(tempStr)))
    return hasil

def addMat2D(mat1,mat2):
    if len(mat1)==len(mat2) and len(mat1[0])==len(mat2[0]):
        hasil=[]
        for i in range(len(mat1)):
            temp=[]
            for j in range(len(mat2)):
                temp.append(mat1[i][j]+mat2[i][j])
            hasil.append(temp)
        return hasil
    else:
        return 'Ukuran tidak sama'
```

In [ ]:
```
mat1=createMat2D(2,3)
```

In [ ]:
```
print(mat1)
```

Kembali ke Menu Awal

Module

Python juga menyediakan fasilitas untuk membuat **Module**. Module ini merupakan suatu file yang terdiri dari sebuah atau lebih dari satu **function**. Dengan mengelompokkan function-function dalam suatu file, akan memudahkan programmer untuk membaca program dan merubah atau memperbaharui program.

Modul ini dapat dipanggil di file lain dengan menggunakan keyword **import**. Terdapat dua buah pilihan, antara lain :

```
import filename
from fileName import *
```

Perbedaan kedua perintah import tersebut adalah pemanggilan function yang terdapat di dalam modul.
Pada pilihan pertama, untuk memanggil fungsi yang ada di modul ini maka menggunakan perintah filename.namaFungsi Sedangkan pada pilihan kedua, untuk memanggil fungsi yang ada di modul ini maka menggunakan perintah namaFungsi

Hanya saja untuk memanggil fungsi-fungsi yang terdapat pada modul tersebut, program pemanggil harus berada di directory yang sama.
Oleh karena itu perlu ditambahkan path pada program pemanggil

Code

Berikut contoh pemanggilan module yang sudah dibuat yaitu checkPrime. Module ini dapat dilihat pada folder code

In [ ]:
```
import checkPrime
```

In [ ]:
```
checkPrime.isPrime(2)
```

Latihan - 3

Buatlah module yang berisi fungsi-fungsi yang terdapat pada Latihan 2, dan gunakanlah modul tersebut untuk mengalikan dua buah matriks

Kembali ke Menu Awal