Buku Pendamping Praktikum Pemrograman Komputer I

Muhammad Khudzaifah
Hisyam Fahmi
Mohammad Jamhuri

March 8, 2018

Contents

Kata Pengantar

Puji syukur ke hadirat Allah SWT, yang telah memberikan rahmat-Nya sehingga Modul Praktikum Pemrograman Komputer I untuk mahasiswa/i Jurusan Matematika Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim ini dapat diselesaikan dengan sebaik-baiknya.

Modul praktikum ini dibuat sebagai pedoman dalam melakukan kegiatan praktikum Modul Praktikum Pemrograman Komputer I yang merupakan kegiatan penunjang mata kuliah Modul Pemrograman Komputer I pada Jurusan Matematika Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim.

Modul praktikum ini diharapkan dapat membantu mahasiswa/i dalam mempersiapkan dan melaksanakan praktikum dengan lebih baik, terarah, dan terencana. Pada setiap topik telah ditetapkan tujuan pelaksanaan praktikum dan semua kegiatan yang harus dilakukan oleh mahasiswa/i serta teori singkat untuk memperdalam pemahaman mahasiswa/i mengenai materi yang dibahas.

Penyusun menyakini bahwa dalam pembuatan Modul Praktikum Pemrograman Komputer I ini masih jauh dari sempurna. Oleh karena itu penyusun mengharapkan kritik dan saran yang membangun guna penyempurnaan modul praktikum ini dimasa yang akan datang. Akhir kata, penyusun mengucapkan banyak terima kasih kepada semua pihak yang

telah membantu baik secara langsung maupun tidak langsung.

Malang, 18 Januari 2018

Penyusun

Part I

Intro, Variabel, List, Fungsi
(4 Pertemuan)

Chapter 1

Pendahuluan

Bahasa pemrograman Python mulai populer saat dikarenakan berbagai hal; mu-dah dipelajari, tersedia dan banyak library-nya. Nanti akan kita bahas bebe-rapa library Python ini. Lengkapnya library ini juga yang menyebabkan Python dipergunakan di berbagai aplikasi. Berbagai sekolah (dan perguruan tinggi) ba-hkan mengajarkan Python sebagai pengantar pemrograman.

Bahasa Python tersedia untuk berbagai sistem operasi; Windows, Mac OS, dan berbagai variasi dari UNIX (Linux, *BSD, dan seterusnya). Di dalam buku ini saya akan menggunakan contoh-contoh yang saya gunakan di komputer saya yang berbasis Windows. Meskipun seharusnya semuanya kompatibel dengan berbagai sistem operasi, kemungkinan ada hal-hal yang agak berbeda. Jika hal itu terjadi, gunakan internet untuk mencari jawabannya.

1.1 Instalasi

Python dapat diperoleh secara gratis dari berbagai sumber. Sumber utamanya adalah di situs python.org. Untuk sementara ini bagian ini saya serahkan kepada Anda. Ada terlalu banyak perubahan sehingga bagian ini akan cepat kadaluwarsa.

Untuk sistem operasi berbasis Linux dan Mac OS, Python sudah terpasang sebagai bawaan dari sistem operasinya. Jika Anda ingin menggunakan versi terbaru maka Anda harus memasangnya sendiri dengan mengunduh instalasinya di python.org.

1.2 Memulai

Sekarang kita dapat memulai pemrograman Python dengan menuliskan program "hello world" (yang merupakan standar bagi belajar pemrograman). Ketikkan "print ..." (dan seterusnya seperti di bawah ini).

```
print("Hello, world!")
Hello, world!
```

Python akan menampilkan apapun yang ada di antara tanda petik tersebut. Hore! Anda berhasil membuat program Python yang pertama.

Mari kita lanjutkan dengan membuat program yang lebih panjang. Program Python dapat disimpan di dalam sebuah berkas untuk kemudian dieksekusi belakangan. Buka editor kesukaan Anda dan ketikkan program hello world di atas di dalam editor Anda tersebut. Setelah itu simpan berkas tersebut dengan nama "hello.py". Biasanya berkas program Python ditandai dengan akhiran (extension) ".py".

Setelah berkas tersebut tersedia, maka kita dapat menjalankan Python dengan memberikan perintah Run Module (F5).

```
Hello, world!
```

1.3 Bahasa Python

Tentang bahasa Python itu sendiri akan diperdalam pada versi berikutnya. Sementara itu fitur tentang bahasa Python akan dibahas sambil berjalan. Pendekatan ini saya ambil untuk membuat buku menjadi lebih menarik dan lebih singkat. Belajar seperlunya.

Hal yang sangat berbeda dari bahasa Python dengan bahasa pemrograman lainnya adalah masalah *block* dari kode. Bahasa pemrograman C misalnya menggunakan tanda kurung kurawal "{" untuk menyatakan blok. Sementara itu Python menggunakan *indentation* untuk menyatakan satu blok. Lihat contoh di bawah ini.

```
for i in range(10):
        print(i)
```

Disarankan untuk menggunakan spasi sebanyak empat (4) buah untuk *indentation* tersebut. (Ini membuat banyak perdebatan karena ada banyak orang yang menggunakan tab bukan spasi.)}

Chapter 2

Variabel

2.1 Tipe Numerik

Python mendukung beberapa tipe data untuk keperluan penyimpanan data numerik. Data numerik yang dapat digunakan meliputi bilangan bulat, bilangan riil, dan bilangan kompleks. Semua objek dari tipe numerik tidak dapat diubah nilainya atau bersifat immutable. Bagian ini menjelaskan tentang masing-masing tipe data tersebut.

2.1.1 Bilangan Bulat

Terdapat dua tipe data bilangan bulat yang didukung oleh python 3, yaitu integer(int) dan boolean(bool). Python 3 tidak memiliki nilai maksimum untuk tipe int. untuk mengonversi bilangan bulat ke string, gunakan fungsi str( ), seperti berikut:

Sebaliknya jika anda ingin mengonversi nilai dari tipe string ke int, gunakan fungsi int( ).

Algorithm 2.1 str( )

```
>>> a=12345
>>> type(a)
<class 'int'>
>>> b=str(a)
>>> b '12345'
>>> type(b)
<class 'str'>
>>>
```

Algorithm 2.2 int( )

```
>>> a='12345'
>>> type(a)
<class 'str'>
>>> b=int(a)
>>> b 12345
>>> type(b)
<class 'int'>
>>>
```

Algorithm 2.3

```
>>> int(True)
1
>>> int(False)
0
>>> a=True
>>> type(a)
<class 'bool'>
>>> int(a)
1
>>>
```

Tipe bool digunakan untuk menyatakan tipe logika (boolean). objek dari tipe bool hanya dapat diisi dengan nilai True atau False (huruf T dan F harus di tulis dalam huruf besar). jika di konversikan ke tipe int, nilai true akan menghasilkan nilai 1 dan False menghasilkan nilai 1.

2.1.2 Bilangan Riil

untuk merepresentasikan data bertipe bilangan riil (mengandung angka di belakang koma), Python menyediakan tipe float. Bilangan dengan tipe float ditulis menggunakan tanda titik

(.), seperti berikut:

Anda juga dapat menulis bilangan riil dalam bentuk eksponen, seperti berikut:

Notasi diatas menunjukkan nilai 8.9 x10^{-4}

untuk mengonversi bilangan dengan tipe float ke string, gunakan fungsi str( ). Sebaliknya, untuk mengonversi string ke tipe float, gunakan fungsi float( ).

Algorithm 2.4

```
>>> a=123.456
>>> a
123.456
>>> type(a)
<class 'float'>
>>> a*2
246.912
>>>
```

Algorithm 2.5

```
>>> a=8.9e-4
>>> a
0.00089
>>>
```

Algorithm 2.6

```
>>> a=123.456
>>> type(a)
<class 'float'>
>>> b=str(a)
>>> b
'123.456'
>>> type(b)
<class 'str'>
>>> c=float(b)
>>> c
123.456
>>> type(c)
<class 'float'>
>>>
```

Algorithm 2.7

```
>>> a=-9+17j
>>> a
(-9+17j)
>>> type(a)
<class 'complex'>
>>> a.real
-9.0
>>> a.imag
17.0
>>>
```

2.1.3 Bilangan Kompleks

bilangan kompleks adalah bilangan yang mengandung pasangan bilangan dari tipe float. Bagian pertama merupakan bagian riil dan bagian kedua merupakan bagian imajiner, kedua bagian tersebut digabung menggunakan tanda + atau - dan diakhiri dengan huruf j.

2.2 Tipe String

Dalam Python, teks (string) merupakan kumpulan karakter Unicode yang direpresentasikan dengan tipe str. Objek string dapat dibuat melalui tiga cara, yaitu:

- Menggunakan tanda petik tunggal
- Menggunakan tanda petik ganda
- Menggunakan tanda petik tunggal atau petik ganda yang ditulis tiga kali

Cara Terakhir biasanya hanya digunakan untuk membuat string panjang yang jumlahnya lebih dari satu baris

Algorithm 2.8

```
>>> s1='PyQt'
>>> s1
'PyQt'
>>> s2="Python"
>>> s2
'Python'
>>> s3=''' Pemrograman GUI dengan Python dan PyQt '''
>>> s3 '\nPemrograman GUI\ndengan Python dan PyQt\n'
>>> s4=""" Pemrograman GUI dengan Python dan PyQt """
>>> s4
'\nPemrograman GUI\ndengan Python dan PyQt\n'
>>> b='python'
>>> b
'python'
>>> b=b.capitalize()
>>> b
'Python'
>>> b=b.upper()
>>> b
'PYTHON'
>>> b=b.lower()
>>> b
'python'
>>> b.isupper()
False
>>> b.islower()
True
>>> s=' '.join(['saya','makan','ayam'])
>>> s
'saya makan ayam'
>>> nim='16610021'
>>> nim.isnumeric()
True >>>
nim='16610o21'
>>> nim.isnumeric()
False
```

Chapter 3

Tipe Koleksi

3.1 Tipe List

List merupakan objek yang bersifat mutable atau nilainya dapat diubah. Kita dapat menambah, mengubah, maupun menghapus elemen-elemen yang terdapat di dalam list. Objek list dibuat menggunakan tanda [ ]. Setiap objek atau elemen yang terdapat di dalam list harus dibatasi menggunakan tanda koma, tapi tidak harus sejenis. Artinya, bisa saja list berisi beberapa objek yang berasal dari tipe berlainan, misalnya str, int, dan sebagainya.

3.2 Tipe Dictionary

Dictionary (kamus) atau sering juga disebut tipe mapping merupakan objek yang berisi daftar pasangan kunci dan nilai (key-value pair). Pada struktur data list, elemen-elemen diindeks berdasarkan bilangan positif maupun negatif tergantung dari arah mana elemen-elemen tersebut akan

Algorithm 3.1

```
>>> list1=[100,200,300,400]
>>> list2=[1,'Pemrograman Komputer',12000.00]
>>> list=[10,8,12,6,15]
>>> list
[10, 8, 12, 6, 15]
>>> len(list)#Menghitung banyaknya elemen didalam list
5
>>> li=[10,8,12,6,15]
>>> li
[10, 8, 12, 6, 15]
>>> li[0],li[1],li[2],li[3],li[4]
(10, 8, 12, 6, 15)
>>> li[-5],li[-4],li[-3],li[-2],li[-1]
(10, 8, 12, 6, 15)
>>> li.append(20)#menambahkan elemen dalam list
>>> li.append(25)
>>> li [10, 8, 12, 6, 15, 20, 25]
>>> li.extend([100,200,300])#menambahkan list dalam sebuah list
>>> li
[10, 8, 12, 6, 15, 20, 25, 100, 200, 300]
>>> li[0]=99
>>> li[1]=77
>>> li
[99, 77, 12, 6, 15, 20, 25, 100, 200, 300]
>>> li.remove(99) #menghapus elemen di dalam list
>>> li.remove(300)
>>> li.remove(15)
>>> li
[77, 12, 6, 20, 25, 100, 200]
>>> li.clear() #Menghapus Semua elemen didalam list
>>> li
[]
```

Algorithm 3.2

```
>>> na={'A':4,'B':3,'C':2,'D':1,'E':0}
>>> na
{'A': 4, 'B': 3, 'C': 2, 'D': 1, 'E': 0}
>>> na.keys() #Menampilkan Kata Kunci(key) dalam Dictionary
dict_keys(['A', 'B', 'C', 'D', 'E'])
>>> na.values() #Menampilkan Kata Value dalam Dictionary
dict_values([4, 3, 2, 1, 0])
>>> na['A']
4
>>> na['B']
3
>>> na['C']
2
>>>
>>> kamus={'mouse':'tikus','cat':'kucing'}
>>> kamus['cat']
'kucing'
>>> kamus.keys()
dict_keys(['mouse', 'cat'])
>>> kamus.values()
dict_values(['tikus', 'kucing'])
```

diakses. Pada struktur dat dictionary, elemen-elemen akan diindeks berdasarkan kuncinya. Objek yang dijadikan sebagai kunci dapat berasal dari tipe apa saja, tapi pada umumnya berupa string, atau paling tidak berupa bilangan. Berbeda dengan list, dictionary dibuat menggunakan { }. Setiap pasangan kunci dan nilai harus dipisahkan menggunakan tanda (:).

3.3 Tipe Tuple

Tuple adalah tipe koleksi yang mirip dengan list. Pebedaannya, tuple bersifat immutable atau elemen-elemennya tidak dapat diubah, baik nilainya

Algorithm 3.3

```
>>> t=(10,20,30)
>>> t[0]
10
>>> t[1]
20
>>> t[2]
30
>>> t[-3],t[-2],t[-1]
(10, 20, 30)
>>> len(t) #Menghitung banyaknya elemen didalam Tuple
3
```

maupun jumlah elemennya. Ini berarti bahwa kita tidak dapat menambah, mengubah, atau menghapus elemen di dalam tuple. Dengan kata lain, tuple merupakan koleksi yang bersifat konstan. Tuple dibuat menggunakan tanda ( ).

3.4 Tipe Set

set (himpunan) adalah tipe koleksi yang setiap elemennya bersifat unik. Dengan demikian, di dalam set tidak akan pernah ada duplikasi nilai elemen. Jika pada saat pembuatan set terdapat beberapa elemen yang nilainya sama, maka elemen-elemen tersebut hanya akan diambil satu, sisanya secara otomatis akan dibuang. Set dibuat menggunakan fungsi set( ) dengan parameter bisa berupa list, dictionary, tuple, maupun string.

Algorithm 3.4

```
>>> s=set([10,10,20,30,30,30])
>>> s
{10, 20, 30}
>>> len(s)
3
>>> s.add(60) #Menambahkan anggota himpunan(set)
>>> s
{10, 20, 30,60}
>>> len(s)
4
```

Chapter 4

Fungsi

Fungsi adalah bagian atau blok program yang berisi satu tugas spesifik. Ketika dipanggil, fungsi ada yang menghasilkan atau mengembalikan nilai dan ada juga yang tidak. Nilai yang dihasilkan oleh fungsi disebut dengan istilah nilai balik (return value). Dalam beberapa bahasa pemrograman lain, fungsi dengan nilai balik disebut fungsi dan fungsi tanpa nilai balik disebut prosedur. fungsi hanya perlu didefinisikan satu kali, tapi dapat digunakan atau dipanggil berkali-kali. Dalam Phyton, fungsi didefinisikan menggunakan perintah def melalui bentuk umum berikut :

def NamaFungsi(parameter1,parameter2,...):

#badan fungsi

Daftar parameter dari suatu fungsi bersifat opsional, tapi sebagian besar fungsi pada umumnya memiliki satu parameter atau lebih. Terdapat empat jenis fungsi yang dapat dibuat di dalam Phyton, yaitu : fungsi global, fungsi lokal, dan fungsi lambda.

Algorithm 4.1

```
>>> def kali(a,b):
                c=a*b
                return c # mengembalikan nilai ke baris pemanggil

>>> def tulis(s):
                print(s)

>>> z=kali(10,5)
>>> z
50
>>> tulis('Pemrograman Komputer I dengan Python')
Pemrograman Komputer I dengan Python
>>> tulis(z)
50
```

4.1 Fungsi Global

Fungsi global adalah fungsi yang didefinisikan di dalam suatu modul dan dapat dipanggil oleh fungsi lain, baik yang berada di dalam modul yang sama maupun modul lain.

4.2 Fungsi Lokal

Fungsi lokal adalah fungsi yang didefinisikan di dalam fungsi lain. fungsi lokal sering disebut fungsi bersarang (nested function). berbeda dengan fungsi global, fungsi lokal hanya akan dikenal oleh fungsi luar tempat fungsi lokal tersebut didefinisikan.

Algorithm 4.2

```
>>> def persentase(a,b,c):
                def hitungPersen(x):
                        total=a+b+c
                        return(x*100.0)/total
                print("Persentase: %f\t%f\t%f" %
                (hitungPersen(a),hitungPersen(b),hitungPersen(c)))

>>> persentase(50,50,50)
Persentase: 33.333333   33.333333      33.333333
>>> persentase(30,90,30)
Persentase: 20.000000   60.000000      20.000000
```

Algorithm 4.3

```
>>> maks = lambda a,b: a if a>b else b
>>> maks(20,10)
20
>>> maks(100,200)
200
>>>
```

4.3 Fungsi lambda

Fungsi lambda adalah suatu ekspresi untuk menangani tugas-tugas pemrograman yang sederhana. Fungsi jenis ini sering dikenal dengan fungsi tanpa nama (anonymous function) dan dibuat menggunakan kata kunci lambda. Bentuk umum penggunaan kata kunci lambda adalah sebagai berikut :

lambda DaftarParameter: ekspresi

Berikut ini contoh kode yang menunjukkan penggunaan kata kunci lamda.

Jika ditulis dalam bentuk fungsi normal, kode di atas dapat diubah men-

Algorithm 4.4

```
>>> def maks(a,b):
                if a>b:
                                return a
                else:
                                return b

>>> maks(20,10)
20
>>> maks(100,200)
200
```

jadi seperti berikut:

Terdapat beberapa hal yang tidak dapat dilakukan pada fungsi lambda, sehingga kita perlu menggunakan fungsi normal. jika anda ingin menyertakan struktur pengulangan for maupun while, perintah-perintah non-ekspresi, dan perintah yang berjumlah lebih dari satu, maka anda tidak dapat menggunakan fungsi lambda. Fungsi lambda biasanya diperankan sebagai parameter dari suatu fungsi lain. Fungsi yang berperan sebagai parameter sering disebut dengan fungsi callback.

4.4 Parameter Default

Suatu parameter fungsi dapat memiliki nilai default pada saat fungsi tersebut didefinisikan. Proses pengisisan nilai ke dalam parameter dilakukan menggunakan operator penugasan (=). Melalui cara ini, kita dapat memanggil fungsi tanpa menyertakan nilai untuk parameter bersangkutan. Parameter seperti ini sering disebut parameter default atau parameter opsional. Parameter default harus ditempatkan pada urutan paling akhir dari daftar parameter lain.

Algorithm 4.5

```
>>> def tulis(s, gantibaris=True):
                if not gantibaris:
                        print(s,end='')
                else:
                        print(s)

>>> tulis('Python')
Python
>>> tulis('Python',False);tulis(' dan Ruby')
Python dan Ruby
>>> tulis('Python');tulis('Ruby')
Python
Ruby
>>>
```

Parameter gantibaris pada contoh kode di atas merupakan parameter default. Nilai default untuk parameter tersebut adalah True. Ini berarti bahwa jika fungsi tulis( ) dipanggil tanpa menyertakan parameterkedua, maka parameter gantibaris akan diisi dengan nilai True.

Part II

Control Flow, Looping, Visualization
(4 Pertemuan)

Modul 4
Control Flow

Tujuan Khusus Praktikum

Mahasiswa dapat menggunakan pernyataan pemilihan untuk membuat program dalam Python.

Ekspresi Boolean

Sebuah ekspresi Boolean adalah ekspresi yang bernilai *true* atau *false*. Ekspresi ini digunakan untuk membandingkan dua nilai atau variabel (operand). Contoh berikut merupakan ekspresi boolean menggunakan operator == (sama dengan/*equality*) untuk membandingkan 2 buah nilai:

```
>>> 5 == 5
True
>>> 5 == 6
False
```

`True` dan `False` merupakan tipe data khusus, yaitu **boolean**, bukan tipe data *string*. Selain operator **==**, ada beberapa operator lain yang bisa digunakan untuk membuat ekspresi boolean, operator-operator ini disebut operator relasi. Operator relasi ditunjukkan pada Tabel 1.

Table 1: Operator Relasi

Ekspresi	Deskripsi
`x == y`	bernilai *True* jika `x` sama dengan `y`
`x != y`	bernilai *True* jika `x` tidak sama dengan `y`
`x > y`	bernilai *True* jika `x` lebih dari `y`
`x < y`	bernilai *True* jika `x` kurang dari `y`
`x >= y`	bernilai *True* jika `x` lebih dari atau sama dengan `y`
`x <= y`	bernilai *True* jika `x` kurang dari atau sama dengan `y`

Logical Operators

Operator logika merupakan jenis operator yang akan membandingkan logika hasil dari operator relasi. Terdapat 3 macam operator yang termasuk dalam operator logika yaitu: `and`, `or`, dan `not`. Deskripsi dari masing-masing operator tersebut dijelaskan pada Tabel 2.
Contoh operator `and` adalah `x > 0 and x < 10` akan bernilai `True` jika dan hanya jika `x` lebih besar dari 0 dan kurang dari 10. Ekspresi `n%2 == 0 or n%3 == 0` akan bernilai `True` jika bilangan `n` habis dibagi 2 **atau** 3. Ekspresi `not (x > y)` bernilai `True` jika `x > y` bernilai `False`, yaitu jika `x` kurang dari atau sama dengan `y`.

Table 2: Operator Logika

Operator	Deskripsi
and	Melakukan pengecekan kondisi yang harus bernilai *True* untuk kedua operand kanan dan kiri secara bersamaan
or	Melakukan pengecekan kondisi yang dapat bernilai *True* pada salah satu atau kedua operand kanan dan kiri
not	Melakukan pengecekan kondisi NOT, atau membalikkan kondisi. Contoh NOT(A AND B)

Conditional execution

Pada saat membuat sebuah program, kadangkala kita membutuhkan pemilihan pernyataan mana yang akan dijalankan oleh komputer. Kemampuan pemilihan perintah inilah yang disebut *conditional statement*. Contoh sederhana dari pernyataan *conditional* ini adalah sebagai berikut:

```
if x > 0:
    print('x is positive')
```

Pada contoh di atas, pernyataan `print 'x is positive'` akan dijalankan jika `x > 0`, sedangkan jika sebaliknya maka pernyataan tersebut tidak dijalankan. Ada beberapa *conditional statement* yang akan dibahan pada modul ini.

Conditional if

Conditional if digunakan untuk memilih apakah sebuah pernyataan akan dijalankan atau tidak sesuai kondisi yang diberikan. Alur pemilihan `if` ditunjukkan pada Gambar 1.

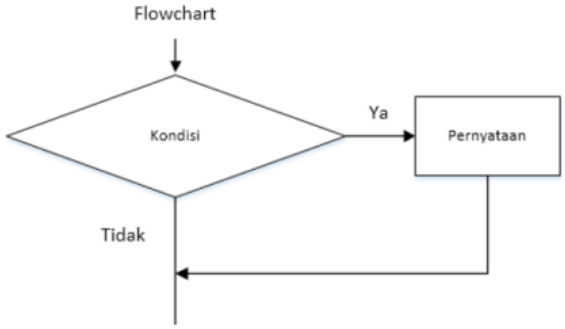

Figure 1: Alur Pemilihan if

Sintaks pemilihan `if` adalah sebagai berikut:

```
if kondisi:
   statement
```

Kondisi pada sintaks tersebut dapat berisi ekspresi relasi dan atau ekspresi logika.

Conditional if-else

Conditional if-else digunakan untuk memilih pernyataan mana yang akan dijalankan dari 2 pernyataan sesuai kondisi yang diberikan. Alur pemilihan `if-else` ditunjukkan pada Gambar 2.

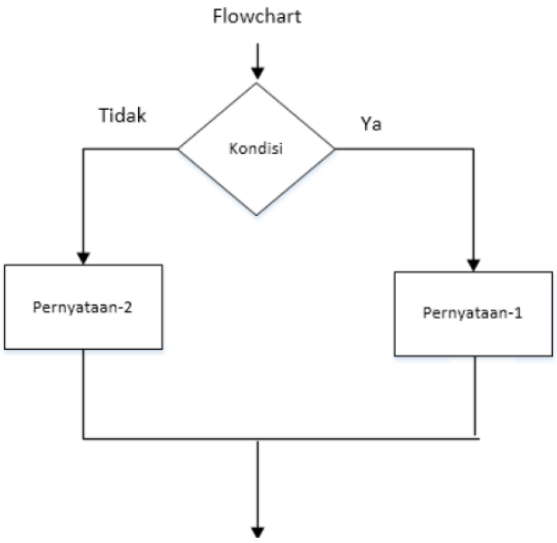

Figure 2: Alur Pemilihan if-else

Sintaks pemilihan `if-else` adalah sebagai berikut:

```
if kondisi:
    statement1
else:
    statement2
```

Pernyataan pada blok `if` akan dijalankan jika **kondisi** bernilai `True`, tapi jika bernilai `False` maka pernyataan pada blok `else` akan dijalankan.

Conditional if-elif-else

Conditional if-elif-else digunakan untuk memilih pernyataan mana yang akan dijalankan dengan beberapa kondisi pengecekan. Alur pemilihan `if-elif-else` ditunjukkan pada Gambar 3.
Sintaks pemilihan `if-elif-else` adalah sebagai berikut:

```
if kondisi1:
    statement1
elif kondisi2:
    statement2
:
else:
    statementX
```

Pernyataan pada blok `if` akan dijalankan jika **kondisi1** bernilai `True`, tapi jika bernilai `False` maka akan dicek **kondisi2** pada `elif` dan seterusnya sampai dengan `else`.

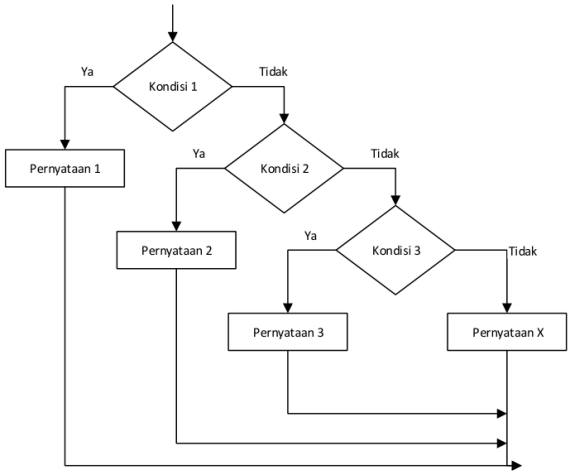

Figure 3: Alur Pemilihan if-elif-else

Langkah Praktikum

Percobaan *Conditional if*

1. Buka *python editor* dan buat *script* baru dengan nama "Percobaan1.py".
2. Tuliskan kode untuk mengambil masukan dari *user*.

```
nilai = input("Masukkan nilai Anda:  ")
```

3. Tuliskan kode *conditional if* untuk melakukan pengecekan nilai. Jangan lupa konversikan variabel `nilai` menjadi `int`.

```
if int(nilai) >= 70:
    print("Anda lulus ujian!  SELAMAT!")
```

4. Jalankan program tersebut dan perhatikan hasilnya.

Percobaan *Conditional if-else*

1. Buka *script* Percobaan1.py yang sudah Anda buat.
2. Tambahkan kode *conditional if-else* untuk melakukan pengecekan nilai apakah lulus atau tidak.

```
if int(nilai) >= 70:
    print("Anda lulus ujian!  SELAMAT!")
else:
    print("Anda belum lulus ujian!")
```

3. Jalankan program tersebut dan perhatikan hasilnya.

Percobaan *Conditional if-elif-else*

1. Buka *script* Percobaan1.py yang sudah Anda buat pada Percobaan 1 dan 2.

2. Tambahkan kode *conditional if-elif-else* untuk melakukan pengecekan nilai apakah valid dan lulus atau tidak.

```
if int(nilai) < 0 or int(nilai) > 100:
    print("Nilai Anda TIDAK valid")
elif int(nilai) >= 70:
    print("Anda lulus ujian!  SELAMAT!")
else:
    print("Anda belum lulus ujian!")
```

3. Jalankan program tersebut dan perhatikan hasilnya.

Latihan

1. Buatlah program untuk meng*input*kan dua buah bilangan bulat, kemudian mencetak salah satu bilangan yang nilainya terbesar!

2. Pada akhir semester seorang dosen menghitung nilai akhir dari mahasiswa yang terdiri dari nilai uas, uts, kuis, dan tugas. Nilai akhir didapatkan dari 40% nilai uas, 30% nilai uts, 10% nilai kuis, dan 20% nilai tugas. Jika nilai akhir dari mahasiswa di bawah 65 maka mahasiswa tersebut akan mendapatkan remidi. Buatlah program untuk membantu mengetahui mahasiswa yang mendapatkan remidi berdasarkan nilai akhir yang didapatkannya!

3. Buatlah program kalkulator sederhana menggunakan Python. *User* akan memasukkan dua buah bilangan riil dan satu buah operator aritmatika (+, -, *, atau /), kemudian program akan mengoperasikan dua bilangan tersebut dengan operator yang sesuai. Contoh tampilan program:

```
>>>
Masukkan bilangan pertama: 5
Masukkan operator (+,-,*,/): *
Masukkan bilangan kedua: 2.5
5.0 * 2.5 = 12.5
>>>
```

Modul 6
Repitition

Tujuan Khusus Praktikum

Mahasiswa dapat menggunakan pernyataan perulangan untuk membuat program dalam Python.

Looping

Loop adalah suatu blok atau kumpulan instruksi yang dilaksanakan secara berulang-ulang. Perulangan yang disebut juga *repetition* akan membuat efisiensi proses dibandingkan jika dioperasikan secara manual. Diagram pada Gambar 1 mengilustrasikan sebuah pernyataan *loop*.

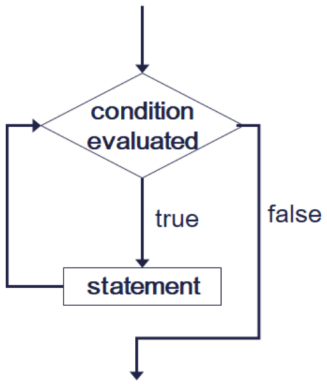

Figure 1: Alur Perulangan

Ada dua jenis perulangan dalam Python, yaitu `while` *loop* dan `for` *loop*.

While Loop

Perulangan dengan `while` akan mengulang sebuah pernyataan atau kumpulan pernyataan jika kondisi yang diberikan bernilai `True`. Kondisi akan dicek terlebih dahulu sebelum menjalankan *body loop*. Sintaks perulangan `while` adalah sebagai berikut:

```
while kondisi:
    statement
```

Contoh penggunaannya adalah sebagai berikut:

```
count = 0
while count < 5:
    print(count)
    count += 1 #menampilkan 0 1 2 3 4
```

For Loop

Perulangan dengan `loop` melakukan perulangan setiap elemen pada sebuah kumpulan data (array, list, dictionary, range). Sintaks perulangan `for` adalah sebagai berikut:

```
for element in sequence:
    statement
```

Contoh penggunaannya adalah sebagai berikut:

```
primes = [2,3,5,7]
for prime in primes:
    print(prime) #menampilkan 2 3 5 7
```

Langkah Praktikum

Percobaan *While Loop*

Program untuk menghitung nilai faktorial.

1. Buka *python editor* dan buat *script* baru dengan nama "Faktorial.py".
2. Tuliskan fungsi `faktorial` yang berisi perulangan dengan *while* untuk menghitung nilai faktorial.

   ```
   def faktorial(n):
       fac = 1
       i = 1
       while i <= n:
           fac = fac*i
           i += 1
       return fac
   ```

3. Tuliskan kode untuk mengambil masukan dari *user*. Jangan lupa konversikan variabel `n` menjadi `int`.

   ```
   angka = int(input("Masukkan nilai yang akan dihitung:  "))
   ```

4. Jalankan program tersebut dan perhatikan hasilnya.

   ```
   faktorial(angka)
   ```

Percobaan *For Loop*

Program perulangan dengan `for` untuk menghitung jumlah total.

1. Buka *python editor* dan buat *script* baru dengan nama "ForLoop.py".
2. Buat list yang berisi beberapa angka.

   ```
   numbers = [1,10,20,30,40,50]
   ```

3. Tuliskan kode untuk melakukan perhitungan jumlah total angka yang ada dalam list.

```
sum = 0
for number in numbers:
    sum += number
```

4. Jalankan program tersebut dan tampilkan hasilnya.

```
print(sum)
```

Percobaan *Looping* dengan *Break*

1. Buka *python editor* dan buat *script* baru dengan nama "LoopBreak.py".
2. Tuliskan kode program berikut:

```
b = 0
while True:
    angka = int(input("Masukkan Angka:  "))
    b += angka
    if b > 50:
        break print("Angka berhenti pada jumlah:  " + b)
```

3. Jalankan program tersebut dan perhatikan hasilnya.

Latihan

1. Buatlah program yang meminta masukan *user* sebuah bilangan bulat `N` dimana (`N > 0`). Program kemudian menampilkan penjumlahan `N` bilangan genap positif pertama (bilangan genap ≥ 0). Contoh:
   - Jika *user* memasukkan `N = 3`, maka *output*nya: `0 + 2 + 4 = 6`
   - Jika *user* memasukkan `N = 5`, maka *output*nya: `0 + 2 + 4 + 6 + 8 = 20`
2. Buatlah sebuah program yang meminta masukan *user* sebuah bilangan bulat `N` dimana (`N > 0`). Kemudian, program menampilkan penjumlahan `N` bilangan kuadrat pertama. Bilangan kuadrat adalah $= 1, 4, 9, 16, 25, 36, ..., N^2$. Contoh:
   - Jika *user* memasukkan `N = 2`, maka *output*nya: `1 + 4 = 5`
   - Jika *user* memasukkan `N = 3`, maka *output*nya: `1 + 4 + 9 = 14`
3. Buatlah sebuah program yang meminta masukan *user* sebuah bilangan bulat `N` dimana (`N > 0`). Program kemudian memeriksa setiap digit yang ada di angka tersebut, dan menampilkan berapa jumlah digit yang ganjil dari bilangan `N` tersebut.
   - Jika *user* memasukkan `N = 2345`, jumlah digit yang ganjil = `2`
   - Jika *user* memasukkan `N = 993312`, jumlah digit yang ganjil = `5`

Modul 6
Data Visualization

Tujuan Khusus Praktikum

Mahasiswa dapat melakukan visualisasi data pada Python.

Visualisasi Data

Python sudah sangat mendukung untuk visualisasi data baik untuk *plot* grafik, *scatter*, *bar*, maupun yang lain. Ada banyak library yang bisa dipakai untuk melakukan visualisasi data, seperti `matplotlib`, `bokeh`, `seaborn`, dan sebagainya. Namun, tidak semua library kita bahas dalam tutorial ini.

Matplotlib

Visualisasi data pada python dapat dilakukan menggunakan library `matplotlib`. `Matplotlib` merupakan salah satu *library* python untuk *plotting* grafik 2D dengan environment yang interaktif. Untuk visualisasi sederhana kita bisa menggunakan modul `pyplot` pada library `matplotlib`. Visualisasi data menggunakan `matplotlib` langkah-langkah utamanya adalah:

1. *import library* `matplotlib` dan *library* lain yang dibutuhkan
2. menyiapkan data
3. *plot* data
4. menampilkan *plot* grafik

Contoh membuat plot grafik sederhana menggunakan `matplotlib` adalah sebagai berikut:

```
# import library
import matplotlib.pyplot as plt

# persiapan data
x = [1,2,3,4]
y = [10,20,25,30]

# membuat plot grafik
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(x, y, color='lightblue', linewidth=3)
ax.scatter([2,4,6],[5,15,25],color='darkgreen',marker='*')
ax.set_xlim(1, 6.5)

# menyimpan grafik
plt.savefig('foo.png')

# menampilkan grafik
plt.show()
```

Persiapan Data

Tahapan ini adalah tahapan untuk menyiapkan data, baik data 1D maupun data 2D. Persiapan data dapat dilakukan dengan membaca dari *file* (*import file*) atau dengan men*generate* data dengan program.

1D Data

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 10, 100)
y = np.cos(x)
z = np.sin(x)
```

2D Data atau Gambar

```
data = 2 * np.random.random((10, 10))
data2 = 3 * np.random.random((10, 10))
Y, X = np.mgrid[-3:3:100j, -3:3:100j]
U = -1 - X**2 + Y
V = 1 + X - Y**2
from matplotlib.cbook import get_sample_data img =
np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

Import Spreadsheet

```
import pandas as pd
# Load csv
df = pd.read_csv('example.csv')

# Load excel spreadsheet
xl = pd.ExcelFile('example.xlsx')
# Print sheet names
print(xl.sheet_names)
# Load sheet ke sebuah DataFrame dengan nama:  df1
df1 = xl.parse('Sheet1')
```

Membuat Plot

```
fig = plt.figure()
fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

Figure dan Axes

- **Figure** adalah seluruh *window* atau halaman. Kita dapat membuat banyak *figure*. Sebuah *figure* dapat mengandung beberapa komponen di dalamnya, seperti *title*, *legend*, *axes*, dan sebagainya.

- **Axes** adalah area di mana data akan divisualisaikan menggunakan fungsi `plot()` atau `scatter()`. Di dalam *figure* bisa terdapat banyak *axes*.

```
fig.add_axes()
ax1 = fig.add_subplot(221) # row-col-num
ax3 = fig.add_subplot(212)
fig3, axes = plt.subplots(nrows=2,ncols=2)
fig4, axes2 = plt.subplots(ncols=3)
```

Melakukan Plot

Inisialisasikan *figure* dan *axes* yang akan dipakai

```
fig, ax = plt.subplots()
```

Menggambar titik dengan garis atau tanda yang menghubungkan masing-masing titik

```
lines = ax.plot(x,y)
```

Menggambar titik yang terpisah, berbeda ukuran dan atau warnanya

```
ax.scatter(x,y)
```

Menggambar diagram batang vertikal

```
axes[0,0].bar([1,2,3],[3,4,5])
```

Menggambar diagram batang horisontal

```
axes[1,0].barh([0.5,1,2.5],[0,1,2])
```

Menggambar garis horisontal pada *axes*

```
axes[1,1].axhline(0.45)
```

Menggambar garis vertikal pada *axes*

```
axes[0,1].axvline(0.65)
```

Menggambar *polygon* tertutup warna tertutup

```
ax.fill(x,y,color='blue')
```

Mewarnai antara y =nilai dan 0

```
ax.fill_between(x,y,color='yellow')
```

Menampilkan dan Menyimpan Plot

Menampilkan *plot* grafik

```
plt.show()
```

Menyimpan *figure*

```
plt.savefig('foo.png')
# Menyimpan figure dengan background transparan
plt.savefig('foo.png', transparent=True)
```

Latihan

Part III

Numpy, Scipy, dan Panda
(4 Pertemuan)

Chapter 5

Praktikum 9: Pendahuluan NumPy

Tujuan Praktikum

Mahasiswa dapat membuat, memodifikasi, mengoperasikan dan menampilkan `array` di *Python Shell*.

5.1 Pendahuluan

Topik utama dari *NumPy* adalah array multidimensi yang diindeks oleh tuple atau pasangan bilangan bulat positif. Dalam *NumPy* dimensi disebut sebagai *axis*, dan banyaknya *axis* disebut *rank*. Sebagai contoh, koordinat sebuah titik diruang 3D dinyatakan sebagai `[1,2,1]` yang merupakan sebuah *array* dengan *rank* 1 karena terdiri dari satu *axis*. Sedangkan panjang *axis* tersebut adalah 3.

Pada contoh berikut, *array*-nya mempunyai *rank* 2. *Axis* pertama panjangnya 2, dan *axis* kedua panjangnya 3.

```
[[ 1., 0., 0.],
 [ 0., 1., 2.]]
```

array *NumPy* disebut sebagai `ndarray`, atau dikenal juga dengan sebutan `array`. Catat bahwa `numpy.array` tidak sama dengan array Standard Python `array.array`, yang hanya bisa digunakan untuk array satu dimensi dan dengan fitur yang lebih sedikit. Atribut yang paling penting dari `ndarray` adalah:

`ndarray.ndim` banyaknya axis atau dimensi dari array. Di Python, banyaknya dimensi disebut sebagai `rank`.

`ndarray.shape` dimensi dari array. Ini adalah pasangan bilangan bulat yang menunjukkan ukuran array di setiap dimensi. Untuk matriks dengan `n` baris dan `m` kolom, bentuknya adalah `(n, m)`. Panjang dari pasangan `shape` adalah `rank`, atau jumlah dimensi, `ndim`.

`ndarray.size` jumlah elemen dari array. Ini sama dengan perkalian elemen dari `shape`.

`ndarray.dtype` sebuah objek yang menggambarkan jenis elemen dalam array. Seseorang dapat membuat atau menentukan jenis dtype menggunakan jenis Python standar. Selain itu NumPy menyediakan jenisnya sendiri. `numpy.int32`, `numpy.int16`, dan `numpy.float64` adalah beberapa contohnya.

`ndarray.itemsize` ukuran dalam byte dari setiap elemen array. Sebagai contoh, sebuah array dari elemen tipe `float64` memiliki `itemsize` 8 (= 64/8), sedangkan satu dari tipe `complex32` memiliki `itemsize` 4 (= 32/8). Ini sama dengan `ndarray.dtype.itemsize`.

`ndarray.data` buffer yang mengandung elemen sebenarnya dari array. Biasanya, kita tidak perlu menggunakan atribut ini karena kita

akan mengakses elemen dalam array menggunakan fasilitas pengindeksan.

Contoh:

```
>>> import numpy as np
>>> a = np.arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> a.shape
(3, 5)
>>> a.ndim
2
>>> a.dtype.name
'int64'
>>> a.itemsize
8
>>> a.size
15
>>> type(a)
<type 'numpy.ndarray'>
>>> b = np.array([6, 7, 8])
>>> b
array([6, 7, 8])
>>> type(b)
<type 'numpy.ndarray'>
```

5.2 Membuat Array

Ada beberapa cara untuk membuat *array*. Misalnya, Anda dapat membuat *array* dari *list* Python biasa atau pasangan menggunakan fungsi `array`. Tipe *array* yang dihasilkan disimpulkan dari tipe elemen dalam array.

```
>>> import numpy as np
>>> a = np.array([2,3,4])
>>> a
array([2, 3, 4])
>>> a.dtype
dtype('int64')
>>> b = np.array([1.2, 3.5, 5.1])
>>> b.dtype dtype('float64')
```

Kesalahan yang sering terjadi dalam pembuatan `array` diantaranya adalah:

```
>>> a = np.array(1,2,3,4)    # WRONG
>>> a = np.array([1,2,3,4])  # RIGHT
```

`array` mengubah barisan dari sebuah barisan menjadi `array` dua dimensi, barisan dari barisan sebuah barisan menjadi `array` tiga dimensi, dan seterusnya.

```
>>> b = np.array([(1.5,2,3), (4,5,6)])
>>> b
array([[ 1.5,  2. ,  3. ],
       [ 4. ,  5. ,  6. ]])
```

Jenis `array` juga dapat secara eksplisit ditentukan pada saat pembuatan:

```
>>> c = np.array( [ [1,2], [3,4] ], dtype=complex )
>>> c
array([[ 1.+0.j,  2.+0.j],
       [ 3.+0.j,  4.+0.j]])
```

Seringkali, elemen dari sebuah `array` pada awalnya tidak diketahui, namun ukurannya diketahui. Oleh karena itu, *NumPy* menawarkan beberapa fungsi untuk membuat `array` yang memuat nilai awal. Hal tersebut dapat meminimalkan pertumbuhan memori yang diperlukan `array`, sebagai operasi yang dianggap mahal.

Fungsi `zeros` menciptakan sebuah `array` yang berisi angka nol, fungsi `ones` yang menciptakan `array` yang berisi angka satu, dan fungsi `empty` men-

ciptakan array yang isinya acak dan bergantung pada keadaan memori. Secara default, `dtype` dari `array` yang dibuat adalah `float64`.

```
>>> np.zeros( (3,4) )
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])
>>> np.ones( (2,3,4), dtype=np.int16 )    # dtype can also be
    specified
array([[[ 1, 1, 1, 1],
        [ 1, 1, 1, 1],
        [ 1, 1, 1, 1]],
       [[ 1, 1, 1, 1],
        [ 1, 1, 1, 1],
        [ 1, 1, 1, 1]]], dtype=int16)
>>> np.empty( (2,3) )    # uninitialized, output may vary
array([[  3.73603959e-262,   6.02658058e-154,   6.55490914e
    -260],
       [  5.30498948e-313,   3.14673309e-307,   1.00000000e
          +000]])
```

Untuk membuat barisan angka, *NumPy* menyediakan fungsi yang mirip dengan range yang outputnya berupa array dan bukan berupa list.

```
>>> np.arange(10,30,5 )
array([10, 15, 20, 25])
>>> np.arange(0,2,0.3 )    # it accepts float arguments
array([ 0. ,  0.3,  0.6,  0.9,  1.2,  1.5,  1.8])
```

Jika arange digunakan dengan argumen floating point, biasanya sulit memprediksi jumlah elemen yang diperoleh, karena presisi floating point yang terbatas. Untuk itu, lebih baik menggunakan fungsi linspace yang dapat menerima masukan berapa jumlah elemen yang kita inginkan:

```
>>> from numpy import pi
>>> np.linspace(0,2,9)    #9 numbers from 0 to 2
array([0.,0.25,0.5,0.75,1.,1.25,1.5,1.75,2.])
```

```
>>> x = np.linspace(0,2*pi,100 )    #useful to evaluate
    function at lots of points
>>> f = np.sin(x)
```

5.3 Menampilkan Array

Saat kita menampilkan array, NumPy menampilkannya dengan cara yang mirip dengan daftar bersarang, namun dengan tata letak berikut:

- axis terakhir dicetak dari kiri ke kanan,
- yang kedua sampai yang terakhir dicetak dari atas ke bawah,
- Sisanya juga dicetak dari atas ke bawah, dengan masing-masing potongan dipisahkan oleh sebuah baris kosong.

array satu dimensi dicetak sebagai baris, array dua dimensi sebagai matriks dan tridimensionals sebagai daftar matriks.

```
>>> a = np.arange(6)    #1d array
>>> print(a)
[0 1 2 3 4 5]
>>>
>>> b = np.arange(12).reshape(4,3)    # 2d array
>>> print(b)
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
>>> >>> c = np.arange(24).reshape(2,3,4)    '''3d array'''
>>> print(c)
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]
```

```
 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```

Jika array terlalu besar untuk ditampilkan, *NumPy* secara otomatis memotong bagian tengah array dan hanya mencetak ujung-ujungnya:

```
>>> print(np.arange(10000))
[   0    1    2 ..., 9997 9998 9999]
>>>
>>> print(np.arange(10000).reshape(100,100))
[[   0    1    2 ...,   97   98   99]
 [ 100  101  102 ...,  197  198  199]
 [ 200  201  202 ...,  297  298  299]
 ...,
 [9700 9701 9702 ..., 9797 9798 9799]
 [9800 9801 9802 ..., 9897 9898 9899]
 [9900 9901 9902 ..., 9997 9998 9999]]
```

Untuk memaksa *NumPy* mencetak seluruh bagian array, kita dapat mengubah opsi print menggunakan set_printoptions.

```
>>> np.set_printoptions(threshold='nan')
```

Chapter 6

Praktikum 10: Operasi Dasar, dan Fungsi Transenden

Tujuan Praktikum

6.1 Operasi-operasi Dasar pada *NumPy*

Operator aritmatika pada `array` menggunakan aturan *elementwise*, yaitu operasi diterapkan elemen per elemen. Sebuah array baru dibuat dan diisi dengan hasilnya.

```
>>> a = np.array( [20,30,40,50] )
>>> b = np.arange( 4 )
>>> b
array([0, 1, 2, 3])
>>> c = a-b
>>> c
array([20, 29, 38, 47])
>>> b**2
array([0, 1, 4, 9])
>>> 10*np.sin(a)
```

```
array([ 9.12945251, -9.88031624,  7.4511316 , -2.62374854])
>>> a<35
array([ True, True, False, False], dtype=bool)
```

Tidak seperti dalam banyak bahasa matriks, operator kali * mengoperasikan perkalian elemen per elemen antar array NumPy. Sedangkan perkalian matriks dapat dilakukan dengan menggunakan fungsi `dot` atau cara sebagaimana berikut:

```
>>> A = np.array( [[1,1],[0,1]] )
>>> B = np.array( [[2,0],[3,4]] )
>>> A*B   # elementwise product
array([[2, 0], [0, 4]])
>>> A.dot(B)   # matrix product
array([[5, 4], [3, 4]])
>>> np.dot(A, B)   # another matrix product
array([[5, 4], [3, 4]])
```

Beberapa operasi, seperti `+=` dan `*=`, berguna untuk memodifikasi `array` yang ada bukan untuk membuat yang baru.

```
>>> a = np.ones((2,3), dtype=int)
>>> b = np.random.random((2,3))
>>> a *= 3
>>> a
array([[3, 3, 3], [3, 3, 3]])
>>> b += a
>>> b
array([[3.417022,3.72032449,3.00011437],
       [ 3.30233257,3.14675589,3.09233859]])
>>> a += b   # b is not automatically converted to integer
   type Traceback (most recent call last):
...
TypeError: Cannot cast ufunc add output from dtype('float64')
    to dtype('int64') with casting rule 'same_kind'
```

Ketika mengoperasikan array dari berbagai tipe, tipe yang dihasilkan adalah tipe yang lebih umum (perilaku ini dikenal sebagai *upcasting*).

```
>>> a = np.ones(3, dtype=np.int32)
>>> b = np.linspace(0,pi,3)
>>> b.dtype.name
'float64'
>>> c = a+b
>>> c
array([ 1.        ,  2.57079633,  4.14159265])
>>> c.dtype.name
'float64'
>>> d = np.exp(c*1j)
>>> d
array([ 0.54030231+0.84147098j, -0.84147098+0.54030231j,
          -0.54030231-0.84147098j])
>>> d.dtype.name
'complex128'
```

Operasi yang tidak biasa, seperti menghitung jumlah semua elemen dalam array, diimplementasikan sebagai kelas `ndarray`.

```
>>> a = np.random.random((2,3))
>>> a
array([[ 0.18626021,  0.34556073,  0.39676747],
       [ 0.53881673,  0.41919451,  0.6852195 ]])
>>> a.sum()
2.5718191614547998
>>> a.min()
0.1862602113776709
>>> a.max()
0.6852195003967595
```

Secara default, operasi yang diberlakukan pada array adalah seperti pada *list* yang berisi bilangan, terlepas dari bentuknya. Namun, dengan menentukan parameter sumbu, Anda dapat menerapkan operasi sepanjang sumbu yang ditentukan dari sebuah array:

```
>>> b = np.arange(12).reshape(3,4)
>>> b
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>>
>>> b.sum(axis=0)    # sum of each column
array([12, 15, 18, 21])
>>>
>>> b.min(axis=1)    # min of each row
array([0, 4, 8])
>>>
>>> b.cumsum(axis=1)    # cumulative sum along each row
array([[ 0,  1,  3,  6],
       [ 4,  9, 15, 22],
       [ 8, 17, 27, 38]])
```

6.2 Fungsi Universal

NumPy menyediakan fungsi matematika yang familiar seperti sin, cos, dan exp. Dalam NumPy, ini disebut "fungsi universal" (ufunc). Dalam NumPy, fungsi ini beroperasi secara elementer pada array, menghasilkan array sebagai output.

```
>>> B = np.arange(3)
>>> B
array([0, 1, 2])
>>> np.exp(B)
array([ 1.        ,  2.71828183,  7.3890561 ])
>>> np.sqrt(B)
array([ 0.        ,  1.        ,  1.41421356])
>>> C = np.array([2., -1., 4.])
>>> np.add(B, C)
array([ 2.,  0.,  6.])
```

Coba juga: `all`, `any`, `apply_along_axis`, `argmax`, `argmin`, `argsort`, `average`, `bincount`, `ceil`, `clip`, `conj`, `corrcoef`, `cov`, `cross`, `cumprod`, `cumsum`, `diff`, `dot`, `floor`, `inner`,

inv, lexsort, max, maximum, mean, median, min, minimum, nonzero, outer, prod,
re, round, sort, std, sum, trace, transpose, var, vdot, vectorize, where.

Chapter 7

Praktikum 11: Modifikasi dan Manipulasi Array

Tujuan Praktikum

Mahasiswa dapat memodifikasi dan memanipulasi `array` *NumPy*.

7.1 Indexing, Slicing and Iterating

Array satu-dimensi dapat diindeks, diiris dan diiterasi, seperti `list` dan jenis barisan lain pada Python.

```
>>> a = np.arange(10)**3
>>> a
array([  0,   1,   8,  27,  64, 125, 216, 343, 512, 729])
>>> a[2]
8
>>> a[2:5]
array([ 8, 27, 64])
```

```
>>> a[:6:2] = -1000    # equivalent to a[0:6:2] = -1000; from
    start to position 6, exclusive, set every 2nd element to
    -1000
>>> a
array([-1000,     1, -1000,    27, -1000,   125,   216,
   343,   512,   729])
>>> a[ : :-1]                                    # reversed a
array([  729,   512,   343,   216,   125, -1000,    27,
   -1000,     1, -1000])
>>> for i in a:
...     print(i**(1/3.))
...
nan
1.0
nan
3.0
nan
5.0
6.0
7.0
8.0
9.0
```

Array multidimensi dapat memiliki satu indeks per axis. Indeks ini diberikan dalam tuple yang dipisahkan koma:

```
>>> def f(x,y):
...     return 10*x+y
...
>>> b = np.fromfunction(f,(5,4),dtype=int)
>>> b
array([[ 0,  1,  2,  3],
       [10, 11, 12, 13],
       [20, 21, 22, 23],
       [30, 31, 32, 33],
       [40, 41, 42, 43]])
>>> b[2,3]
```

```
23
>>> b[0:5, 1]    # each row in the second column of b
array([ 1, 11, 21, 31, 41])
>>> b[ : ,1]     # equivalent to the previous example
array([ 1, 11, 21, 31, 41])
>>> b[1:3, : ]   # each column in the second and third row of
    b
array([[10, 11, 12, 13],
       [20, 21, 22, 23]])
```

Jika banyak indeks yang dimasukkan lebih sedikit dari dimensi array, maka indeks yang tidak ditulis dianggap sebagai irisan lengkap:

```
>>> b[-1]        # the last row. Equivalent to b[-1,:]
array([40, 41, 42, 43])
```

Ekspresi dengan tanda kurung dalam `b[i]` dianggap sebagai `i` diikuti oleh `:` yang diperlukan untuk mewakili dimensi yang tersisa. *NumPy* juga membolehkan kita untuk menulis ini menggunakan titik sebagai `b[i, ...]`.

Titik-titik `(...)` mewakili banyak titik dua yang diperlukan untuk menghasilkan pasangan pengindeksan yang lengkap. Misalnya, jika `x` adalah array 5 dimensi, maka

- `x[1,2,...]` ekivalen dengan `x[1,2,:,:,:]`,
- `x[...,3]` sampai `x[:,:,:,:,3]` dan
- `x[4,...,5,:]` sampai `x[4,:,:,5,:]`.

```
>>> c = np.array( [[[  0,  1,  2],       # a 3D array (two
    stacked 2D arrays)
...                 [ 10, 12, 13]],
...                [[100,101,102],
...                 [110,112,113]]])
>>> c.shape
(2, 2, 3)
```

```
>>> c[1,...]     # same as c[1,:,:] or c[1]
array([[100, 101, 102],
       [110, 112, 113]])
>>> c[...,2]     # same as c[:,:,2]
array([[  2,  13],
       [102, 113]])
```

Iterasi atas array multidimensi dilakukan sehubungan dengan sumbu pertama:

```
>>> for row in b:
...     print(row)
...
[0 1 2 3]
[10 11 12 13]
[20 21 22 23]
[30 31 32 33]
[40 41 42 43]
```

Namun, jika seseorang ingin melakukan operasi pada setiap elemen dalam array, kita dapat menggunakan atribut datar yang merupakan `iterator` atas semua elemen dari array:

```
>>> for element in b.flat:
...     print(element)
...
0
1
2
3
10
11
12
13
20
21
22
23
```

```
30
31
32
33
40
41
42
43
```

7.2 Manipulasi Bentuk

7.2.1 Mengubah bentuk sebuah array

Array memiliki bentuk yang diberikan oleh jumlah elemen sepanjang masing-masing sumbu:

```
>>> a = np.floor(10*np.random.random((3,4)))
>>> a
array([[ 2.,  8.,  0.,  6.],
       [ 4.,  5.,  1.,  1.],
       [ 8.,  9.,  3.,  6.]])
>>> a.shape
(3, 4)
```

Bentuk array bisa diubah dengan berbagai perintah. Perhatikan bahwa tiga perintah berikut semua mengembalikan array yang dimodifikasi, namun jangan mengubah array aslinya:

```
>>> a.ravel()    # returns the array, flattened
array([ 2.,  8.,  0.,  6.,  4.,  5.,  1.,  1.,  8.,  9.,  3.,
     6.])
>>> a.reshape(6,2)       # returns the array with a modified
   shape
array([[ 2.,  8.],
       [ 0.,  6.],
```

```
       [ 4.,  5.],
       [ 1.,  1.],
       [ 8.,  9.],
       [ 3.,  6.]])
>>> a.T # returns the array, transposed
array([[ 2.,  4.,  8.],
       [ 8.,  5.,  9.],
       [ 0.,  1.,  3.],
       [ 6.,  1.,  6.]])
>>> a.T.shape
(4, 3)
>>> a.shape
(3, 4)
```

Urutan elemen dalam array yang dihasilkan dari ravel () biasanya "C-style", yaitu indeks paling kanan "berubah paling cepat", jadi elemen setelah [0,0] adalah [0,1] . Jika array dibentuk kembali ke bentuk lain, lagi-lagi array diperlakukan sebagai "Gaya C". NumPy biasanya membuat array yang tersimpan dalam urutan ini, jadi ravel () biasanya tidak perlu menyalin argumennya, tapi jika array dibuat dengan mengambil irisan array lain atau dibuat dengan opsi yang tidak biasa, mungkin perlu disalin. Fungsi ravel () dan reshape () juga dapat diinstruksikan, dengan menggunakan argumen opsional, untuk menggunakan array gaya FORTRAN, di mana indeks paling kiri berubah paling cepat.

Fungsi reshape mengembalikan argumennya dengan bentuk yang dimodifikasi, sedangkan metode ndarray.resize memodifikasi array itu sendiri:

```
>>> a
array([[ 2.,  8.,  0.,  6.],
       [ 4.,  5.,  1.,  1.],
       [ 8.,  9.,  3.,  6.]])
>>> a.resize((2,6))
>>> a
array([[ 2.,  8.,  0.,  6.,  4.,  5.],
       [ 1.,  1.,  8.,  9.,  3.,  6.]])
```

Jika dimensi diberikan sebagai -1 dalam operasi pengubah ukuran, dimensi lainnya dihitung secara otomatis:

```
>>> a.reshape(3,-1)
array([[ 2.,  8.,  0.,  6.],
       [ 4.,  5.,  1.,  1.],
       [ 8.,  9.,  3.,  6.]])
```

7.3 Menumpuk bersama array yang berbeda

Beberapa array dapat ditumpuk bersama-sama di sepanjang sumbu yang berbeda:

```
>>> a = np.floor(10*np.random.random((2,2)))
>>> a
array([[ 8.,  8.],
       [ 0.,  0.]])
>>> b = np.floor(10*np.random.random((2,2)))
>>> b
array([[ 1.,  8.],
       [ 0.,  4.]])
>>> np.vstack((a,b))
array([[ 8.,  8.],
       [ 0.,  0.],
       [ 1.,  8.],
       [ 0.,  4.]])
>>> np.hstack((a,b))
array([[ 8.,  8.,  1.,  8.],
       [ 0.,  0.,  0.,  4.]])
```

Fungsi `column_stack` menumpuk array 1D sebagai kolom menjadi array 2D. Ini setara dengan `vstack` hanya untuk array 1D:

```
>>> from numpy import newaxis
>>> np.column_stack((a,b))    # With 2D arrays
array([[ 8.,  8.,  1.,  8.],
```

```
       [ 0.,  0.,  0.,  4.]])
>>> a = np.array([4.,2.])
>>> b = np.array([2.,8.])
>>> a[:,newaxis]  # This allows to have a 2D columns vector
array([[ 4.],
       [ 2.]])
>>> np.column_stack((a[:,newaxis],b[:,newaxis]))
array([[ 4.,  2.],
       [ 2.,  8.]])
>>> np.vstack((a[:,newaxis],b[:,newaxis])) # The behavior of
   vstack is different
array([[ 4.],
       [ 2.],
       [ 2.],
       [ 8.]])
```

Untuk array dengan lebih dari dua dimensi, tumpukan `hstack` di sepanjang sumbu kedua mereka, tumpukan `vstack` di sepanjang sumbu pertama mereka, dan `concatenate` memungkinkan sebuah argumen opsional yang memberi jumlah sumbu yang dengannya penggabungan tersebut harus terjadi.

Catatan

Dalam kasus kompleks, `r_` dan `c_` berguna untuk membuat array dengan menumpuk angka sepanjang satu sumbu. Mereka mengizinkan penggunaan literal jangkauan (":")

```
>>> np.r_[1:4,0,4]
array([1, 2, 3, 0, 4])
```

Bila digunakan dengan array sebagai argumen, `r_` dan `c_` mirip dengan `vstack` dan `hstack` dalam perilaku default mereka, namun izinkanlah argumen opsional yang memberi jumlah sumbu untuk menggabungkannya.

Chapter 8

Praktikum 12: Modifikasi dan Manipulasi Array Lebih Lanjut

Tujuan Praktikum

Mahasiswa dapat memodifikasi dan manipulasi `array` serta dapat menggunakan fungsi-fungsi *NumPy* untuk menyelesaikan masalah-masalah dalam aljabar linier.

8.1 Memecah array menjadi beberapa bagian yang lebih kecil

Dengan menggunakan `hsplit`, Anda dapat membagi sebuah array di sepanjang sumbu horisontalnya, dengan menentukan jumlah array berbentuk sama untuk kembali, atau dengan menentukan kolom setelah pembagian tersebut terjadi:

```
>>> a = np.floor(10*np.random.random((2,12)))
```

```
>>> a
array([[ 9.,  5.,  6.,  3.,  6.,  8.,  0.,  7.,  9.,  7.,
    2.,  7.],
       [ 1.,  4.,  9.,  2.,  2.,  1.,  0.,  6.,  2.,  2.,
         4.,  0.]])
>>> np.hsplit(a,3)   # Split a into 3
[array([[ 9.,  5.,  6.,  3.],
        [ 1.,  4.,  9.,  2.]]), array([[ 6.,  8.,  0.,  7.],
        [ 2.,  1.,  0.,  6.]]), array([[ 9.,  7.,  2.,  7.],
        [ 2.,  2.,  4.,  0.]])]
>>> np.hsplit(a,(3,4))   # Split a after the third and the
    fourth column
[array([[ 9.,  5.,  6.],
        [ 1.,  4.,  9.]]), array([[ 3.],
        [ 2.]]), array([[ 6.,  8.,  0.,  7.,  9.,  7.,  2.,
          7.],
        [ 2.,  1.,  0.,  6.,  2.,  2.,  4.,  0.]])]
```

`vsplit` terbagi sepanjang sumbu vertikal, dan `array_split` memungkinkan seseorang untuk menentukan sepanjang sumbu mana yang akan dipecah.

8.2 Menyalin dan menampilkan

Saat mengoperasikan dan memanipulasi array, datanya terkadang disalin ke array baru dan terkadang tidak. Ini sering menjadi sumber kebingungan bagi pemula. Ada tiga kasus:

Jangan menyalin semuanya

Tugas sederhana tidak membuat salinan objek array atau datanya.

```
>>> a = np.arange(12)
>>> b = a            # no new object is created
```

```
>>> b is a             # a and b are two names for the same
    ndarray object True
>>> b.shape = 3,4      # changes the shape of a
>>> a.shape
(3, 4)
```

Python melewati objek yang bisa berubah sebagai referensi, jadi pemanggilan fungsi tidak membuat salinan.

```
>>> def f(x):
...     print(id(x))
...
>>> id(a)         # id is a unique identifier of an object
148293216
>>> f(a)
148293216
```

Lihat atau Salin Dangkal

Objek array yang berbeda dapat berbagi data yang sama. Metode view menciptakan objek array baru yang melihat data yang sama.

```
>>> c = a.view()
>>> c is a
False
>>> c.base is a # c is a view of the data owned by a
True
>>> c.flags.owndata
False
>>>
>>> c.shape = 2,6         # a's shape doesn't change
>>> a.shape
(3, 4)
>>> c[0,4] = 1234         # a's data changes
>>> a
array([[   0,    1,    2,    3],
```

```
       [1234,    5,    6,    7],
       [   8,    9,   10,   11]])
```

Slicing sebuah array mengembalikan pandangannya:

```
>>> s = a[ : , 1:3]      # spaces added for clarity; could
    also be written "s = a[:,1:3]"
>>> s[:] = 10    # s[:] is a view of s. Note the difference
    between s=10 and s[:]=10
>>> a
array([[   0,   10,   10,    3],
       [1234,   10,   10,    7],
       [   8,   10,   10,   11]])
```

Deep Copy

Metode copy membuat salinan lengkap dari array dan datanya.

```
>>> d = a.copy()          # a new array object with new data is
    created
>>> d is a
False
>>> d.base is a           # d doesn't share anything with a
False
>>> d[0,0] = 9999
>>> a
array([[   0,   10,   10,    3],
       [1234,   10,   10,    7],
       [   8,   10,   10,   11]])
```

8.3 Ringkasan Fungsi dan Metode

Array Creation `arange, array, copy, empty, empty_like, eye, fromfile, fromfunction, identity, linspace, logspace, mgrid, ogrid, ones, ones_like, r, zeros, zeros_like`

Conversions `ndarray.astype, atleast_1d, atleast_2d, atleast_3d, mat`

Manipulations `array_split, column_stack, concatenate, diagonal, dsplit, dstack, hsplit, hstack, ndarray.item, newaxis, ravel, repeat, reshape, resize, squeeze, swapaxes, take, transpose, vsplit, vstack`

Questions `all, any, nonzero, where`

Ordering `argmax, argmin, argsort, max, min, ptp, searchsorted, sort`

Operations `choose, compress, cumprod, cumsum, inner, ndarray.fill, imag, prod, put, putmask, real, sum`

Basic Statistics `cov, mean, std, var`

Basic Linear Algebra `cross, dot, outer, linalg.svd, vdot`

8.4 Aljabar Linier

Operasi-operasi Array

```
>>> import numpy as np
>>> a = np.array([[1.0, 2.0], [3.0, 4.0]])
>>> print(a)
[[ 1.  2.]
 [ 3.  4.]]
>>> a.transpose()
array([[ 1.,  3.],
       [ 2.,  4.]])
>>> np.linalg.inv(a)
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
>>> u = np.eye(2) # unit 2x2 matrix; "eye" represents "I"
>>> u
array([[ 1.,  0.],
       [ 0.,  1.]])
>>> j = np.array([[0.0, -1.0], [1.0, 0.0]])
```

```
>>> np.dot(j, j) # matrix product
array([[-1.,  0.],
       [ 0., -1.]])
>>> np.trace(u)  # trace 2.0
>>> y = np.array([[5.], [7.]])
>>> np.linalg.solve(a, y)
array([[-3.],
       [ 4.]])
>>> np.linalg.eig(j)
(array([0.+1.j,0.-1.j]), array([[ 0.70710678+0.j
   ,0.70710678-0.j],
       [0.00000000-0.70710678j,  0.00000000+0.70710678j]]))
```

Automatic Reshaping

Untuk mengubah dimensi array, Anda dapat menghilangkan salah satu ukuran yang kemudian akan disimpulkan secara otomatis:

```
>>> a = np.arange(30)
>>> a.shape = 2,-1,3  # -1 means "whatever is needed"
>>> a.shape (2, 5, 3)
>>> a
array([[[ 0,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8],
        [ 9, 10, 11],
        [12, 13, 14]],
       [[15, 16, 17],
        [18, 19, 20],
        [21, 22, 23],
        [24, 25, 26],
        [27, 28, 29]]])
```

Penumpukan Vektor

Membentuk array 2D dari beberapa vektor baris di MATLAB dapat dilakukan dengan cara yang cukup mudah, jika `x` dan `y` adalah dua vektor dengan panjang yang sama, kita hanya perlu menuliskan `m = [x; y]`. Dalam *NumPy* hal tersebut dapat dilakukan dengan menggunakan fungsi `column_stack`, `dstack`, `hstack` dan `vstack`, tergantung pada dimensi array yang dikehendaki. Sebagai contoh:

```
x = np.arange(0,10,2)    # x=([0,2,4,6,8])
y = np.arange(5)          # y=([0,1,2,3,4])
m = np.vstack([x,y])      # m=([[0,2,4,6,8],
                          #     [0,1,2,3,4]])
xy = np.hstack([x,y])    # xy =([0,2,4,6,8,0,1,2,3,4])
```

Chapter 9

Praktikum 13: Scipy

Tujuan Praktikum

Mahasiswa dapat menggunakan *SymPy* untuk mengintegralkan, menurunkan, memfaktorkan, melimitkan persamaan-persamaan matematika.

Pengantar SymPy

SymPy sangat berguna dalam melakukan perhitungan sains di *Python*, seperti integral, turunan, interpolasi, limit, fungsi-fungsi transenden dan lain sebagainya.

Sebagai contoh lihat perbedaan perhitungan antara $\sqrt{9}$ dan $\sqrt{8}$ menggunakan modul math dan sympy,

```
import math as mt
import sympy as sy
a = mt.sqrt(9)
b = sy.sqrt(9)
```

```
c = mt.sqrt(8)
d = sy.sqrt(8)

print(a,b)
print(c,d)
```

Print Out dari kode Python diatas adalah

```
3.0 3
2.8284271247461903 2*sqrt(2)
```

Jika menggunakan modul math nilai dari $\sqrt{8} = 2.82\ldots$, sedangkan jika menggunaan modul sympy nilai yang ditampilkan adalah $2\sqrt{2}$.

Pada *SymPy* variabel didefinisikan dengan symbols, sebagai contoh mari kita definisikan persamaan matematika $x + 2y$

```
from sympy import symbols
x,y = symbols('x y')
expr = x + 2*y

print(expr)
print(expr-x)
print(expr**2)
```

Print Out dari kode Python diatas adalah

```
x + 2*y
2*y
(x + 2*y)**2
```

Variabel yang didefinisikan dengan symbols juga dapat dioperasikan dengan bilangan, sebagai contoh lihat kode berikut

```
expr
Out[7]: x + 2*y
expr+1
Out[8]: x + 2*y + 1
x*expr
Out[9]: x*(x + 2*y)
```

SymPy dapat digunakan untuk menyederhanakan persamaan, mengintegralkan, menurunkan, menghitung limitnya, menyelesaikan persamaan, dan masih banyak lagi. Berikut ini beberapa contoh yang dapat dilakukan menggunakan *SymPy*.

```
from sympy import symbols, expand, factor

x,y = symbols('x y')
expr = x + 2*y
p1 = x*expr
p2 = expand(p1)

print('expr = ',expr)
print('p1 =',p1)
print('p2 =',p2)
```

Print out dari kode Python diatas adalah:

```
expr =  x + 2*y
p1 = x*(x + 2*y)
p2 = x**2 + 2*x*y
```

Pada *SymPy* juga ada cara untuk mengubah tampilan dari *Print out* menjadi lebih indah, lihat cara pemakaiannya pada contoh berikut

```
from sympy import *
x, t, z, nu = symbols('x t z nu')
init_printing(use_unicode=True)

p1 = sin(x)*exp(x)
p2 = diff(p1,x)
p3 = sin(x)*exp(x)+exp(x)*cos(x)
p4 = integrate(p3,x)
p5 = integrate(sin(x**2),(x,-oo,oo))
```

Selanjutnya pada *console* dapat kita ketikkan `p1`, `p2`, `p3`, `p4`, ataupun `p5` untuk melihat hasil dari kode *Python* diatas.

```
In [35]: p1,p2
Out[35]:
```

$$(e^x \sin(x), \quad e^x \sin(x) + e^x \cos(x))$$

```
In [36]: p3,p4,p5
Out[36]:
```

$$\left(e^x \sin(x) + e^x \cos(x), \quad e^x \sin(x), \quad \frac{\sqrt{2}\sqrt{\pi}}{2}\right)$$

Beberapa fungsi bawaan *SymPy* yang bisa digunakan diantaranya adalah:

limit limit(sin(x)/x, x, 0)

solve solve(x**2-2, x)

dsolve y = Function('y')
dsolve(Eq(y(t)*diff(t,t)-y(t),exp(t)), y(t))

simplify simplify(sin(x)**2+cos(x)**2)

```
In [44]: (x+1)**3
Out[44]:
```

$$(x+1)^3$$

```
In [45]: expand(In[44])
Out[45]:
```

$$x^3 + 3x^2 + 3x + 1$$

```
In [46]: expand((x+2)*(x-3))
Out[46]:
```

$$x^2 - x - 6$$

```
In [47]: factor(x**3-x**2+x-1)
Out[47]:
```

$$(x-1)\left(x^2+1\right)$$

```
In [48]: simplify(cos(x)**2+sin(x)**2)
Out[48]:
```

$$1$$

Kenapa SymPy

Karena *SymPy* gratis, *open source*, dan berada dibawah lisensi dari BSD sehingga kita bebas untuk memodifikasi, menyebarkan ulang ataupun menjualnya. Untuk informasi lebih lanjut mengenai penggunaan *SymPy* dapat diperoleh pada Tutorial SymPy.