Khoirudin, S.Kom., M.Eng Lahir di Kendal, 22 Februari 1989. Penulis menamatkan pendidikan S1 di Universitas Stikubank Semarang pada tahun 2020 konsentrasi bidang studi Teknik Informatika. Pada tahun 2011-2015 penulis mendapatkan Beasiswa Kedutaan China di Jakarta, Indonesia, untuk belajar Bahasa Mandarin dan menamatkan program S2 Magister Komputer dengan konsentrasi Computer Applied Technology.

Penerbit:
UNIVERSITAS SEMARANG PRESS
Jl. Soekarno-Hatta, Tlogosari, Semarang 50196
Telp: (024) 6702757, Fax: (024) 6702272
E-mail: usmpress@usm.ac.id
http://www.usmpress.usm.ac.id

ISBN 978-602-9019-87-2

USM PRESS

ALGORITMA & STRUKTUR DATA DENGAN PHYTON 3

Khoirudin, S.Kom., M.Eng.

python

ALGORITMA & STRUKTUR DATA DENGAN PHYTON 3

Khoirudin, S.Kom., M.Eng.

JUDUL BUKU
Algoritma dan Struktur Data Dengan *Python* 3

Penulis:
Khoirudin.

ISBN:
978-602-9019-87-2

Desain Grafis:
Desi Eka Sari

Tata Letak:
Diyah Kartika Sari
Sahesti Ningtyas.

Penerbit:

Universitas Semarang Press

Redaksi:
Jl. Soekarno-Hatta Pedurungan Semarang. 50196 Indonesia
Telp: 024-6702757, Fax: 024-6702272
e-mail: usmpress@usm.ac.id
http://www.usmpress.usm.ac.id

Cetakan Pertama, November 2019
vi, 75 hlm, 15.5 cm x 23 cm

KATA PENGANTAR

Alhaamdulillah buku Algoritma dan Struktur Data dengan *Python* 3 ini sudah selesai disusun, buku ini bertujuan memberikan gambaran dasar-dasar dan teori praktis tentang bagaimana mengimplementasikan algoritma dan struktur data dengan Bahasa pemrograman *Python* 3. *Python* sendiri adalah bahasa pemrograman interpretatif multiguna yang masih keturunan dari Bahasa C dengan filosofi perancangan yang berfokus pada tingkat keterbacaan kode. *Python* sendiri ada yang menggunakan Bahasa versi 2 dan Bahasa versi 3, dalam buku ini materi disusun dengan menggunakan *Python* 3.

Penulis mengucapkan terimakasih banyak kepada pihak-pihak yang telah berkontribusi dalam penyusunan buku ini, penulis sadar betul buku ini masih jauh dari sempurna. Oleh sebab itu penulis membuka kritik dan saran guna untuk menyempurnakan buku ini dalam edisi selanjutnya. Harapan penulis buku ini dapat bermanfaat khususnya dalam proses pembelajaran di program studi Teknik Informatika Universitas Semarang.

Semarang, November 2019

Khoirudin

DAFTAR ISI

BAB 1 PENGENALAN *PYTHON*

A. Pengenalan *Python*

Python adalah bahasa pemrograman interpretatif yang multiplatform dan multiguna, *Python* termasuk di dalam bahasa pemrograman tingkat tinggi karya Guido van Rossum. *Python* sendiri telah banyak digunakan untuk membuat berbagai macam aplikasi dan program, seperti: Program GUI (desktop), Aplikasi Mobile, Web, IoT, Game, bahkan Program untuk Hacking.

Dengan keunggulan kode yang simpel dan mudah diimplementasikan, seorang programmer dapat lebih meng-utamakan pengembangan aplikasi yang dibuat, bukan malah sibuk mencari syntax error.

```
print("Python itu sangat simple broo");
```

Dengan hanya menuliskan kode print seperti contoh diatas, kita sudah dapat mencetak apapun yang kita inginkan di dalam tanda kurung **()**. Di bagian akhir kode pun, kita tidak harus meng-akhirnya dengan tanda **semicolon** (**;**).

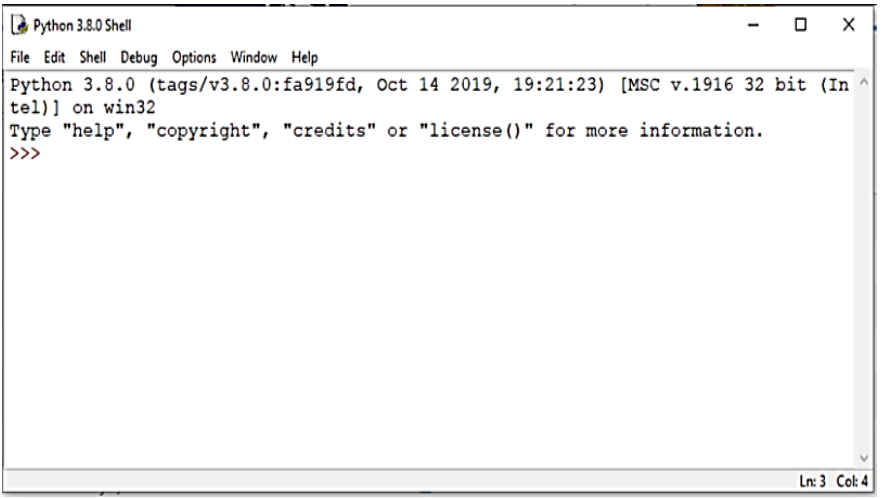
B. Instalasi *Python*

Sebelum kita menggunakan *Python*, kita harus menginstal-nya terlebih dahulu di sistem operasi komputer Anda. Saat ini *Python* memiliki 2 versi yang berbeda, yaitu *Python* versi 3 dan *Python* versi 2, dimana versi terbaru dari *Python* 2 adalah ***Python* 2.7.17** dan untuk versi 3 adalah ***Python* 3.8.0** untuk versi stablenya. Untuk

mendapatkan file instalasi *Python* kita dapat mengunjungi laman berikut

https://www.*Python*.org/downloads/,

dalam buku ini kita akan belajar bahasa pemrograman Python menggunakan versi terbaru **3.8.0**. Halaman **IDE Python 3.8** dapat kita lihat seperti gambar berikut



Gambar 1. IDLE *Python* 3.8.0

C. *Integrated Development Environment* (IDE) *Python*

IDE adalah sebuah software aplikasi yang memberikan kita fasilitas yang sangat bermanfaat ketika membuat aplikasi atau program. Biasanya sebuah IDE terdiri dari *source code editor build automation debugger* dan *tools*. Untuk menulis sebuah program, bisa menggunakan text editor atau IDE nya. Bagi yang sudah mahir, menulis program dengan text editor bukanlah menjadi masalah. Tetapi untuk pemula, akan lebih mudah menggunakan IDE.

Beberapa IDE untuk *Python* yang cukup populer adalah :

1. ***Web Development***: Django, Pyramid, Bottle, Tornado, Flask, web2py.
2. ***GUI Development***: tkInter, PyGObject, PyQt, PySide, Kivy, wx*Python*
3. **Scientific and Numeric**: SciPy, Pandas, IPython
4. ***Software Development***: Buildbot, Trac, Roundup
5. ***System Administration*:** Ansible, Salt, OpenStack

D. Tipe Data *Python*

Tipe data adalah suatu memori atau media pada komputer yang digunakan untuk menampung informasi atau data sementara. *Python* sendiri mempunyai tipe data yang cukup unik bila kita bandingkan dengan bahasa pemrograman yang lain.

Berikut adalah tipe data dari bahasa pemrograman *Python*;

Tabel 1. Tipe Data Python

Tipe Data	Contoh	PenJealasan
`Boolean`	`True atau False`	Menyatakan nilai benar (True) yang bernilai 1, atau nilai salah (False) yang bernilai 0
`String`	`"FTIK USM Joss"`	Menyatakan karakter/kalimat bisa berupa huruf, angka, atau kombinasi keduanya *nilai pada string harus ditulis diantara tanda* **Petik double** *“” atau* **petik tunggal** *‘’*
`Integer`	`13 atau 1234`	Menyatakan bilangan bulat positif
`Float`	`3.14 atau 2.19`	Menyatakan bilangan yang desimal
`Hexadecimal`	`6a atau 2d3`	Menyatakan bilangan dalam format heksa (bilangan berbasis 16)
`Complex`	`1 + 5j`	Menyatakan pasangan angka real dan imajiner
`List`	`['abc', 123, 2.23]`	Data List dapat menyimpan berbagai tipe data dan isinya bisa diubah-ubah
`Tuple`	`('xyz', 768, 2.23)`	Data jenis Tuple yang menyimpan berbagai tipe data tapi isinya tidak bisa diubah
`Dictionary`	`{'nama': 'adi','id':2}`	Data Jenis Dictionary menyimpan berbagai tipe data berupa pasangan penunjuk dan nilai

i

Supaya lebih paham, mari kita coba tipe data dalam Python

```
#tipe data Boolean
print(True)

#tipe data String
print("FTIK USM, Progdi Teknik Informatika")
print('FTIK USM, Progdi Teknik Informatika')

#tipe data Integer
print(80)

#tipe data Float
print(3.14)

#tipe data Complex
print(5j)
```

```
#tipe data List
print([1,2,3,4,5])
print(["satu", "dua", "tiga"])

#tipe data Tuple
print((1,2,3,4,5))
print(("satu", "dua", "tiga"))

#tipe data Dictionary
print({"nama":"Joko", 'umur':22})
#tipe data Dictionary dimasukan ke dalam variabel biodata
biodata = {"nama":"Ani", 'umur':20} #proses inisialisasi variabel biodata
#proses pencetakan variabel biodata yang berisi tipe data Dictionary
print(biodata)
#fungsi untuk mengecek jenis tipe data. akan tampil <class 'dict'>
#yang berarti dict adalah tipe data dictionary
type(biodata)
```

E. Variabel *Python*

Variabel ialah lokasi memori yang dicadangkan guna untuk menyimpan nilai-nilai. Hal ini berarti bahwa saat kita membuat sebuah variabel kita memesan beberapa ruang kosong di dalam memori. Variabel menyimpan data yang proses selama program dieksekusi, yang nantinya isi dari variabel tersebut dapat diubah oleh operasi - operasi tertentu pada program yang menggunakan variable tersebut.

Penulisan variabel *Python* memiliki aturan tertentu, yaitu :

1. Karakter pertama harus berupa huruf atau garis bawah/underscore _
2. Karakter selanjutnya dapat berupa huruf, garis bawah/underscore _ atau angka
3. Karakter pada **nama variabel** bersifat **sensitif** (***case-sensitif***). Artinya huruf kecil dan huruf besar dibedakan. Sebagai contoh, variabel **namaDepan** dan **namadepan** adalah variabel yang berbeda.

```
#proses memasukan data ke dalam variabel
nama = "Universitas Semarang"
#proses mencetak variabel
print(nama)

#nilai dan tipe data dalam variabel  dapat diubah
umur = 20                #nilai awal
print(umur)              #mencetak nilai umur
type(umur)               #mengecek tipe data umur
umur = "dua puluh satu" #nilai setelah diubah
print(umur)              #mencetak nilai umur
type(umur)               #mengecek tipe data umur


namaDepan = "Rudin"
namaBelakang = "Ndeso"
nama = namaDepan + " " + namaBelakang
umur = 29
hobi = "Badminton"
print("Biodata\n", nama, "\n", umur, "\n", hobi)

#contoh variabel lainya
inivariabel = "Halo"
ini_juga_variabel = "Hai"
_inivariabeljuga = "Hi"
inivariabel222 = "Bye"

panjang = 10
lebar = 5
luas = panjang * lebar
print(luas)
```

BAB 2 OPERATOR

A. Operator Aritmatika

Operator	Contoh	Penjelasan
Penjumlahan+	1 + 3 = 4	Menjumlahkan nilai dari masing-masing operan atau bilangan
Pengurangan -	4 - 1 = 3	Mengurangi nilai operan di sebelah kiri menggunakan operan di sebelah kanan
Perkalian *	2 * 4 = 8	Mengalikan operan/bilangan
Pembagian /	10 / 5 = 2	Untuk membagi operan di sebelah kiri menggunakan operan di sebelah kanan
Sisa Bagi%	11 % 2 = 1	Mendapatkan sisa pembagian dari operan di sebelah kiri operator ketika dibagi oleh operan di sebelah kanan
Pangkat **	8 ** 2 = 64	Memangkatkan operan disebelah kiri operator dengan operan di sebelah kanan operator
Pembagian Bulat //	10 // 3 = 3	Sama seperti pembagian. Hanya saja angka dibelakang koma dihilangkan

Operator	Contoh

Penjumlahan+	#Penjumlahan
	print(13 + 2)
	apel = 7
	jeruk = 9
	buah = apel + jeruk #
	print(buah)

Pengurangan -	#Pengurangan
	hutang = 10000
	bayar = 5000
	sisaHutang = hutang - bayar
	print("Sisa hutang kita adalah ", sisaHutang)

Perkalian *	#Perkalian
	panjang = 15
	lebar = 8
	luas = panjang * lebar
	print(luas)

Pembagian /	#Pembagian
	kue = 16
	anak = 4
	kuePerAnak = kue / anak
	print("Setiap anak akan mendapatkan bagian kue sebanyak ", kuePerAnak)

Sisa Bagi%	#Sisa Bagi / Modulus
	bilangan1 = 14
	bilangan2 = 5
	hasil = bilangan1 % bilangan2

Operator	Contoh
	print("Sisa bagi dari bilangan ", bilangan1, " dan ", bilangan2, " adalah ", hasil)

Pangkat **	#Pangkat
	bilangan3 = 8
	bilangan4 = 2
	hasilPangkat = bilangan3 ** bilangan4
	print(hasilPangkat)

Pembagian Bulat //	#Pembagian Bulat
	print(10//3)
	#10 dibagi 3 adalah 3.3333. Karena dibulatkan maka akan menghasilkan nilai 3

B. Operator Perbandingan (*Comparison Operators*)

Operator perbandingan (*comparison operators*) digunakan untuk membandingkan suatu nilai dari masing-masing operan.

Operator	Contoh	Penjelasan
Sama dengan ==	1 == 1 bernilai True	Jika masing-masing operan memiliki nilai yang sama, maka kondisi bernilai benar atau True.
Tidak sama dengan !=	2 != 2 bernilaiFalse	Akan menghasilkan nilai kebalikan dari kondisi sebenarnya.
Tidak sama dengan <>	2 <> 2 bernilaiFalse	Akan menghasilkan nilai kebalikan dari kondisi sebenarnya.
Lebih besar dari >	5 > 3 bernilaiTrue	Jika nilai operan kiri lebih besar dari nilai operan kanan, maka kondisi menjadi benar.
Lebih kecil dari <	5 < 3 bernilaiTrue	Jika nilai operan kiri lebih kecil dari nilai operan kanan, maka kondisi menjadi benar.
Lebih besar atau sama dengan >=	5 >= 3 bernilaiTrue	Jika nilai operan kiri lebih besar dari nilai operan kanan, atau sama, maka kondisi menjadi benar.
Lebih kecil atau sama dengan <=	5 <= 3 bernilaiTrue	Jika nilai operan kiri lebih kecil dari nilai operan kanan, atau sama, maka kondisi menjadi benar.

C. Operator Penugasan (*Assignment Operators*)

Operator penugasan digunakan untuk memberikan atau memodifikasi nilai ke dalam sebuah variabel.

Operator	Contoh	Penjelasan
Sama dengan =	a = 1	Memberikan nilai di kanan ke dalam variabel yang berada di sebelah kiri.
Tambah sama dengan +=	a += 2	Memberikan nilai variabel dengan nilai variabel itu sendiri ditambah dengan nilai di sebelah kanan.
Kurang sama dengan -=	a -= 2	Memberikan nilai variabel dengan nilai variabel itu sendiri dikurangi dengan nilai di sebelah kanan.
Kali sama dengan *=	a *= 2	Memberikan nilai variabel dengan nilai variabel itu sendiri dikali dengan nilai di sebelah kanan.
Bagi sama dengan /=	a /= 4	Memberikan nilai variabel dengan nilai variabel itu sendiri dibagi dengan nilai di sebelah kanan.
Sisa bagi sama dengan %=	a %= 3	Memberikan nilai variabel dengan nilai variabel itu sendiri dibagi dengan nilai di sebelah kanan. Yang diambil nantinya adalah sisa baginya.
Pangkat sama dengan **=	a **= 3	Memberikan nilai variabel dengan nilai variabel itu sendiri dipangkatkan dengan nilai di sebelah kanan.
Pembagian bulat sama dengan //=	a //= 3	Membagi bulat operan sebelah kiri operator dengan operan sebelah kanan operator kemudian hasilnya diisikan ke operan sebelah kiri.

D. Operator Logika (*Logical Operators*)

Operator	Contoh	Penjelasan
and	a, b = True, True # hasil akan True print (a and b)	Jika kedua operan bernilai True, maka kondisi akan bernilai True. Selain kondisi tadi maka akan bernilai False.
or	a, b = True, False # hasil akan True print (a or b) print (b or a) print (a or a) # hasil akan False print (b or b)	Jika salah satu atau kedua operan bernilai True maka kondisi akan bernilai True. Jika keduanya False maka kondisi akan bernilai False.
not	a, b = True, False # hasil akan True print (not a) print (not b)	Membalikkan nilai kebeneran pada operan misal jika asalnya True akan menjadi False dan begitupun sebaliknya.

E. Sample Program

```
print ("************************************************")
print ("    Aplikasi Luas dan Keliling Ruang Bangun    ")
print ("************************************************")

s=float(input("Masukkan sisi persegi = "))
a=float(input("Masukkan alas segitiga = "))
c=float(input("Masukkan sisi miring segitiga = "))
d=float(input("Masukkan sisi miring segitiga = "))
t=float(input("Masukkan tinggi segitiga = "))
b=float(input("Masukkan alas jajar genjang = "))
m=float(input("Masukkan sisi miring sejajar jajar genjang = "))
h=float(input("Masukkan tinggi jajar genjang = "))
L1=s**(2)
L2=(a*t)/2
L3=b*h
K1=4*s
K2=c+d+a
K3=(2*b)+(2*m)
print ()
print ("************************************************")
print ("        Hasil Perhitungan Ruang Bangun          ")
print ("************************************************")

print ("               Luas Ruang Bangun                ")
print ("================================================")
print ("Luas persegi=",L1)
print ("Luas segitiga=",L2)
print ("Luas jajar genjang=",L3)
print ()

print ("           Keliling Ruang Bangun                ")
print ("================================================")
print ("Keliling persegi=",K1)
print ("Keliling segitiga=",K2)
print ("Keliling jajar genjang=",K3)
print ("************************************************")
```

Hasil dari Program sebagai berikut

```
************************************************
    Aplikasi Luas dan Keliling Ruang Bangun
************************************************
Masukkan sisi persegi = 3
Masukkan alas segitiga = 3
Masukkan sisi miring segitiga = 3
Masukkan sisi miring segitiga = 3
Masukkan tinggi segitiga = 3
Masukkan alas jajar genjang = 3
Masukkan sisi miring sejajar jajar genjang = 3
Masukkan tinggi jajar genjang = 3

************************************************
        Hasil Perhitungan Ruang Bangun
************************************************
               Luas Ruang Bangun
================================================

Luas persegi= 9.0
Luas segitiga= 4.5
Luas jajar genjang= 9.0

            Keliling Ruang Bangun
================================================

Keliling persegi= 12.0
Keliling segitiga= 9.0
Keliling jajar genjang= 12.0
************************************************
```

BAB 3 SELEKSI KONDISI

A. Kondisi *If*

Seleksi kondisi (kondisi ***if***) digunakan untuk mengantisi-pasi kondisi yang terjadi saat program dijalankan dan me-nentukan tindakan atau eksekusi apa yang akan diambil sesuai dengan kondisi yang ada. Pada *Python* ada beberapa statement/ kondisi diantaranya adalah ***if, else dan elif,*** kondisi ***if*** digunakan untuk mengeksekusi kode jika kondisi bernilai benar. Jika kondisi bernilai salah maka statement/kondisi ***if*** tidak akan di-eksekusi.

Sample *If* 1:

```
nilai = 9
#jika kondisi benar/TRUE maka program akan mengeksekusi perintah dibawahnya
if(nilai > 7):
    print("Selamat kita Lulus")
#jika kondisi salah/FALSE maka program tidak akan mengeksekusi perintah dibawahnya
if(nilai > 10):
    print("Selamat kita Lulus")
```

B. Kondisi *If Else*

Kondisi ***if else*** adalah kondisi dimana jika pernyataan benar (***true***) maka kode dalam ***if*** akan dieksekusi, tetapi jika bernilai salah (***false***) maka akan mengeksekusi kode di dalam ***else***.

#Sample If Else 1, if.py:

```
nilai = 3
#Jika pernyataan pada if bernilai TRUE maka if
akan dieksekusi, tetapi jika FALSE kode pada else
yang akan dieksekusi.
if(nilai > 7):
   print("Selamat kita Lulus")
else:
   print("Maaf kita Tidak Lulus")
```

#Sample If Else 2, if2.py:

```
kunci="keren"
password=input("masukan password = ")
if password==kunci:
   print ("password Benar")
else:
   print ("password salah")
```

#Sample If Else 3, if3.py:

```
angka = input ("masukan Suatu bilangan = ")
if angka > "0" :
   print ("Bilangan Positif")
elif angka < "0" :
   print ("bilangan Negatif")
else :
   print ("Merupakan 0")
```

#Sample If Else 4, if4.py:

```
i= float (input ("masukan bilangan : "))
if i%2==0:
    if i != 0 :
        print (i,"adalah bilangan genap")
    else :
        print ("angka 0")
else :
    print (" adalah bilangan ganjil")
```

C. Kondisi *Elif*

Seleksi kondisi (kondisi ***if …. elif***) merupakan lanjutan/ percabangan logika dari "***if***". Dengan ***elif*** kita dapat membuat kode program yang akan menyeleksi beberapa kemungkinan yang dapat terjadi. Hal ini hampir sama dengan kondisi "***else*",** bedanya kondisi "***elif***" dapat mempunyai banyak kemungkinan tidak hanya satu.

Sample Elif 1: if5.py

```
hari_ini = "Minggu"

if(hari_ini == "Senin"):
    print("Saya akan kuliah")
elif(hari_ini == "Selasa"):
    print("Saya akan kuliah")
elif(hari_ini == "Rabu"):
    print("Saya akan kuliah")
elif(hari_ini == "Kamis"):
    print("Saya akan kuliah")
elif(hari_ini == "Jumat"):
    print("Saya akan kuliah")
elif(hari_ini == "Sabtu"):
    print("Saya akan kuliah")
elif(hari_ini == "Minggu"):
    print("Saya akan libur")
```

Sample Elif 2: if6.py

```
x = input ("Masukan angka = ")
if x<"5":
    print (x,"lebih kecil dari 5")
elif x=="5":
    print (x,"sama dengan 5")
else:
    print (x,"lebih besar dari 5")
```

BAB 4 PERULANGAN

Secara umum, pernyataan atau program pada bahasa pemrograman akan dieksekusi secara berurutan. Pernyataan pertama dalam sebuah fungsi dijalankan pertama, diikuti oleh yang kedua, dan seterusnya. Akan Tetapi ada situasi dimana kita harus menulis banyak kode, dimana kode tersebut sangat banyak. Hal ini jika kita lakukan secara manual maka kita hanya akan membuang-buang tenaga dengan menulis beratus-ratus bahkan beribu-ribu kode. Untuk itu kita perlu menggunakan Perulangan di dalam bahasa pemrograman *Python*.

A. *While*

Perulangan While Loop di dalam bahasa pemrograman *Python* dieksesusi berkali-kali selama kondisi bernilai benar atau True.

```
#Contoh penggunaan While Loop
hitung = 0
while (hitung < 9):
    print ('Mari berhitung kakak :', hitung)
    hitung = hitung + 1
print ("Sampai Jumpa!")
```

```
Mari berhitung kakak : 0
Mari berhitung kakak : 1
Mari berhitung kakak : 2
Mari berhitung kakak : 3
Mari berhitung kakak : 4
Mari berhitung kakak : 5
Mari berhitung kakak : 6
Mari berhitung kakak : 7
Mari berhitung kakak : 8
Sampai Jumpa!
```

B. *For*

Perulangan *For* pada *Python* memiliki kemampuan untuk mengulangi item dari urutan apapun, seperti list atau string.

```
#Contoh perulangan for sederhana, for.py

angka = [1,2,3,4,5]
for x in angka:
    print(x)
```

Hasil

```
1
2
3
4
5
```

```
#Contoh perulangan for, for2.py

buah = ["nanas", "apel", "jeruk"]
for makanan in buah:
    print("Saya suka makan", makanan)
```

Hasil

```
Saya suka makan nanas
Saya suka makan apel
Saya suka makan jeruk
```

C. Nested

Bahasa pemrograman *Python* memungkinkan penggunaan satu lingkaran di dalam loop lain. Bagian berikut menunjukkan beberapa contoh untuk menggambarkan konsep tersebut.

```
#Contoh penggunaan Nested Loop, while.py

i = 2
while(i < 100):
    j = 2
    while(j <= (i/j)):
        if not(i%j): break
        j = j + 1
    if (j > i/j) : print(i, " FTIK USM")
    i = i + 1

print ("Mari Bergabung Bersama FTIK USM, FTIK..Joss!")
```

Hasil

```
2  FTIK USM
3  FTIK USM
5  FTIK USM
7  FTIK USM
11  FTIK USM
13  FTIK USM
17  FTIK USM
19  FTIK USM
23  FTIK USM
29  FTIK USM
31  FTIK USM
37  FTIK USM
41  FTIK USM
43  FTIK USM
47  FTIK USM
53  FTIK USM
59  FTIK USM
61  FTIK USM
67  FTIK USM
71  FTIK USM
73  FTIK USM
79  FTIK USM
83  FTIK USM
89  FTIK USM
97  FTIK USM
Mari Bergabung Bersama FTIK USM, FTIK Joss!
```

D. Sampel Program

Contoh 1, whileIf.py:

```
nama = "rudin"
kunci = "ndeso"
a=0
while a!=3 :
    username = input ("masukan username : ")
    password = input ("masukan password : ")
    if username == nama and password == kunci :
        print ("password benar")
        break
    elif username == nama or password == kunci:
        print ("user or password error")
    else :
        print ("password salah")
    a=a+1
if a==3:
    print ("sudah 3x input")
```

Hasil

```
masukan username : rudin
masukan password : ok
user or password error
masukan username : rudin
masukan password : ganteng
user or password error
masukan username : rudin
masukan password : ndeso
password benar
```

Contoh 2:

```
#Contoh math.py
import math
r=float(input("jari-jari lingkaran="))
luas=math.pi*r*r
print ("Luas lingkaran=",luas)
```

BAB 5 *NUMBER, STRING,* TANGGAL DAN WAKTU

A. *Number*

Number merupakan tipe data dalam *Python* yang menyimpan nilai berupa angka atau numerik, number juga tipe data yang tidak berubah, ini berarti mengedit nilai dari sejumlah tipe data akan menghasilkan objek yang baru dialokasikan.

Objek Number dibuat saat kita memberikan nilai pada objek tersebut. Sebagai contoh :

```
variabelKesatu = 1
variabelKedua = 11
```

Python mendukung beberapa tipe data *number* diantaranya adalah : `Int, Float, Complex`

Int	Float	Complex
20	0.1	3.14j
300	1.2	35.j
-13	-41.2	3.12e-12j
20	32.23+e123	.873j
-103	-92	-.123+0J
-0x212	-3.25E+11	3e+123J
0x56	60.2-E13	4.31e-4j

1. Konversi Tipe Data *Number*

Pada *Python* kita bisa mengkonversi tipe data dengan menggunakan fungsi. Di bawah ini adalah beberapa fungsi untuk mengkonversi tipe data number *Python*.

- **`int(x)=`** untuk meng-konversi x menjadi plain integer.
- **`long(x)=`** untuk meng-konversi x menjadi long integer.
- **`float(x)=`** untuk meng-konversi x menjadi floating point number.
- **`complex(x)=`** untuk meng-konversi x menjadi complex number dengan real part x dan imaginary part zero.
- **`complex(x, y)=`** untuk meng-konversi x dan y menjadi complex number dengan real part x dan imaginary part y. x dan numeric expressions y.

2. Fungsi Matematika

Nama	Penggunaan	Penjelasan
Absolute	**abs(x)**	Nilai absolut dari x:(positive) jarak antara x and 0.
Ceiling	**ceil(x)**	Ceiling dari x: integer terkecil yang kurang dari x.
Cmp	**cmp(x, y)**	-1 if x < y, 0 if x == y, or 1 if x > y. Tidak berlaku lagi dengan Python 3. Sebaliknya gunakan return (x>y)-(x<y).< td="" style="box-sizing: inherit;"></y).<>
Eksponen	**exp(x)**	Nilai eksponen dari x: ex
Fabs	**fabs(x)**	Nilai absolut dari x.
Floor	**floor(x)**	The floor of x: the largest integer not greater than x.
Floor	**floor(x)**	Nilai dasar dari x: internet terbesar tidak lebih besar dari x.
Log	**log(x)**	Logaritma dari x, untuk x > 0.
Log 10	**log10(x)**	Basis 10 logaritma dari x, untuk x > 0.
Max	**max(x1, x2,...)**	Argumen terbesar: Nilai terdekat dengan tak terhingga positif
Min	**min(x1, x2,...)**	Argumen terkecil: nilai yang paling mendekati tak berhingga negatif.
Modf	**modf(x)**	Bagian pecahan dan bilangan bulat dari x dalam tupel dua item. Kedua bagian memiliki tanda yang sama dengan x. Bagian integer dikembalikan sebagai float.
Pow	**pow(x, y)**	Nilai x ** y.
Round	**round(x [,n])**	X dibulatkan menjadi n digit dari titik desimal. Putaran Python jauh dari nol sebagai tie-breaker: round (0.5) adalah 1.0 dan round (-0.5) adalah -1.0.
Akar Kuadrat	**sqrt(x)**	Akar kuadrat x untuk x> 0.

3. Fungsi Nomor Acak

Fungsi nomor acak digunakan untuk aplikasi-apilkasi permainan, simulasi, pengujian, keamanan, dan privasi. *Python* mencakup fungsi berikut yang umum digunakan. Berikut adalah daftarnya :

Nama	Penggunaan	Penjelasan
Choice	**choice(seq)**	Item acak dari list, tuple, atau string.
RandRange	**randrange ([start,] stop [,step])**	Elemen yang dipilih secara acak dari jangkauan (start, stop, step).
Random	**random()**	A random float r, sehingga 0 kurang dari atau sama dengan r dan r kurang dari 1
Seed	**seed([x])**	Menetapkan nilai awal integer yang digunakan dalam menghasilkan bilangan acak. Panggil fungsi ini sebelum memanggil fungsi modul acak lainnya. Tidak ada pengembalian
Shuffle	**shuffle(lst)**	Mengacak daftar dari daftar di tempat. Tidak ada pengembalian
Floor	**floor(x)**	The floor of x: the largest integer not greater than x.
Uniform	**uniform(x, y)**	Sebuah float acak r, sedemikian rupa sehingga x kurang dari atau sama dengan r dan r kurang dari y.

4. Fungsi Trigonometri

Nama	Penggunaan	Penjelasan
Acos	**acos(x)**	Kembalikan kosinus x, di radian.
Asin	**asin(x)**	Kembalikan busur sinus x, dalam radian.
Atan	**atan(x)**	Kembalikan busur singgung x, di radian.
Atan 2	**atan2(y, x)**	Kembali atan (y / x), di radian.
Kosinus	**cos(x)**	Kembalikan kosinus x radian.
Hypot	**hypot(x, y)**	Kembalikan norma Euclidean, sqrt (x * x + y * y).
Sin	**sin(x)**	Kembalikan sinus dari x radian.
Tan	**tan(x)**	Kembalikan tangen x radian.
Derajat	**degrees(x)**	Mengonversi sudut x dari radian ke derajat.
Radian	**radians(x)**	Mengonversi sudut x dari derajat ke radian.

5. Konstanta Matematika

Nama	Penggunaan	Penjelasan
Pi	pi	Konstanta Pi matematika
e	e	Konstanta e matematika

B. *String*

1. Mengakses Nilai dalam *String*

Python tidak menggunakan tipe karakter titik koma **;** Ini diperlakukan sebagai string dengan panjang satu, sehingga juga dianggap sebagai substring. Untuk mengakses substring, gunakan tanda kurung siku untuk mengiris beserta indeks atau indeks untuk mendapatkan substring Anda. Sebagai contoh :

```
nama = 'FTIK USM'
pesan = "Belajar bahasa Python di FTIK USM"
print ("Nama[0]: ", nama[0])
print ("Pesan [1:4]: ", pesan[1:4])
```

Hasil

```
Nama[0]:  F
Pesan [1:4]:  ela
```

2. Mengupdate *String*

Kita dapat "memperbarui" string yang ada dengan (kembali) menugaskan variabel ke string lain. Nilai baru dapat dikaitkan dengan nilai sebelumnya atau ke string yang sama sekali berbeda sama sekali. Sebagai contoh:

```
pesan = 'FTIK USM'
print ("Perbaharui data tipe string :- ",
pesan[:6] + 'Python')
```

Hasil ➔ `Perbaharui data tipe string :-  FTIK UPython`

3. Operator Spesial *String Python*

Asumsikan variabel string adalah variable a = 'Belajar' dan variabel b adalah = '*Python*', lalu di bawah ini adalah operator yang bisa dipakai pada kedua string di variabel tersebut.

```
a = "Belajar"
b = "Python"
```

Operator	Contoh	Penjelasan
+	**a + b** akan menghasilkan **BelajarPython**	Concatenation - Menambahkan nilai pada kedua sisi operator
*	**a*2** akan menghasilkan **BelajarBelajar**	Perulangan - Membuat string baru, menggabungkan beberapa salinan dari string yang sama
[]	**a[1]** akan menghasilkan **e**	Slice - Memberikan karakter dari indeks yang diberikan
[:]	**a[1:4]** akan menghasilkan **ela**	Range Slice - Memberikan karakter dari kisaran yang diberikan
in	**B in a** akan menghasilkan **1**	Keanggotaan - Mengembalikan nilai true jika ada karakter dalam string yang diberikan
not in	**Z not in a** akan menghasilkan **1**	Keanggotaan - Mengembalikan nilai true jika karakter tidak ada dalam string yang diberikan
r/R	**print r'\n' prints \n dan print R'\n'prints \n**	Raw String - Menekan arti aktual karakter Escape. Sintaks untuk string mentah sama persis dengan string biasa kecuali operator string mentah, huruf "r", yang mendahului tanda petik. "R" bisa berupa huruf kecil (r) atau huruf besar (R) dan harus ditempatkan tepat sebelum tanda kutip pertama.
%		Format - Melakukan format String

4. Operator Format *String Python*

Salah satu fitur *Python* yang paling keren adalah format string operator %. Operator ini unik untuk string dan membuat paket

memiliki fungsi dari keluarga `printf C () C.` berikut adalah contoh sederhananya :

```
print ("My name is %s and weight is %d kg!" %
('Zara', 21))
```

Operator	Penjelasan
%c	character
%s	Konversi string melalui str () sebelum memformat
%i	Dianggap sebagai bilangan bulat desimal
%d	Dianggap sebagai bilangan bulat desimal
%u	Unsigned decimal integer
%o	Bilangan bulat oktal
%x	Bilangan bulat heksadesimal (huruf kecil)
%X	Bilangan bulat heksadesimal (huruf besar)
%e	Notasi eksponensial (dengan huruf kecil 'e')
%E	Notasi eksponensial (dengan huruf besar 'E')
%f	Bilangan real floating point
%g	Yang lebih pendek dari% f dan% e
%G	Lebih pendek dari% f dan% E

5. Triple *Quote Python*

Python **triple quotes** digunakan dengan membiarkan string text untuk ditulis dalam beberapa baris, termasuk kata kerja **NEWLINEs, TABs**, dan karakter khusus lainnya.

Sintaks untuk triple quotes terdiri dari tiga tanda kutip tunggal atau ditulis berturut-turut :

```
kutipantiga = """ ini adalah string panjang yang
terdiri dari beberapa baris dan karakter yang
tidak dapat dicetak seperti TAB (\ t) dan mereka
akan muncul seperti itu ketika ditampilkan.
NEWLINE di dalam string, baik secara eksplisit
diberikan seperti ini di dalam kurung [\ n], atau
```

```
hanya NEWLINE di dalam penugasan variabel juga
akan muncul."""
print (kutipantiga)
```

C. Tanggal dan Waktu

Program *Python* dapat menangani tanggal dan waktu dengan beberapa cara. Konversi antara format tanggal adalah tugas umum untuk komputer. Modul waktu dan kalender *Python* melacak tanggal dan waktu.

1. Apa itu *Tick*?

Interval waktu adalah bilangan floating-point dalam satuan detik. Instansi tertentu dalam waktu dinyatakan dalam hitungan detik sejak pukul 12:00 1 Januari 1970. Dibawah ini adalah contoh penggunanaya.

```
import time;  # Digunakan untuk meng-import modul
time

ticks = time.time()
print ("Berjalan sejak 1.00am, 1 Desember 2019:",
ticks)
```

2. Apa itu TimeTuple *Python*?

Python mempunyai beberapa fungsi yang berguna untuk menangani waktu, seperti pada tabel berikut.

Index	Field	Value
0	4-digit year	2008
1	Bulan	1 sampai 12
2	Hari	1 sampai 31
3	Jam	0 sampai 23
4	Menit	0 sampai 59
5	Detik	0 sampai 61
6	Hari dalam Minggu	0 sampai 6 (0 adalah Senin)
7	Hari dalam Bulan	1 sampai 366
8	Daylight savings	-1, 0, 1, -1 means library determines DST

Fungsi Tupel di atas sama dengan struktur **struct_time**. Struktur ini memiliki atribut sebagai berikut:

Index	Atribut	Value
0	tm_year	2008
1	tm_mon	1 sampai 12
2	tm_mday	1 sampai 31
3	tm_hour	0 sampai 23
4	tm_min	0 sampai 59
5	tm_sec	0 sampai 61
6	tm_wday	0 sampai 6 (0 adalah Senin)
7	tm_yday	1 sampai 366
8	tm_isdst	-1, 0, 1, -1 means library determines DST

3. Mendapatkan Waktu dengan *Python*

Untuk menerjemahkan waktu instan dari satu detik sejak nilai floating-point ke waktu menjadi tupel waktu, lewati nilai floating-point ke fungsi (mis., Localtime) yang mengembalikan waktu tupel dengan semua sembilan item valid.

```
#waktu Saat ini
import time;

localtime = time.localtime(time.time())
print ("Waktu Kita saat ini adalah :", localtime)
```

Kita dapat memformat kapan saja sesuai kebutuhan Anda, namun metode sederhana untuk mendapatkan waktu dalam format yang mudah dibaca adalah asctime ()

```
# Waktu yang berformat
import time;

localtime                                                    =
time.asctime( time.localtime(time.time()) )
print ("Waktu Kita saat ini adalah :", localtime)
```

Modul kalender memberikan berbagai macam metode untuk dimainkan dengan kalender tahunan dan bulanan. Di sini, kita mencoba untuk mencetak kalender bulan tertentu (contoh. Desember 2019)

```
# Kalender Sebulan
import calendar

cal = calendar.month(2019, 12)
print ("Dibawah ini adalah kalender:")
print (cal)
```

4. Modul time

Ada modul waktu populer yang tersedia dengan *Python* yang menyediakan fungsi untuk bekerja dengan waktu dan untuk mengkonversi antara representasi. Dibawah ini adalah tabel dari modul time pada *Python* yang ada.

Fungsi Python	Penjelasan
time.altzone	Diimbangi zona waktu DST lokal, dalam detik di sebelah barat UTC, jika seseorang didefinisikan. Ini negatif jika zona waktu DST lokal berada di sebelah timur UTC (seperti di Eropa Barat, termasuk Inggris). Gunakan saja ini jika siang hari tidak nol.
time.asctime ([tupletime])	Menerima time-tupel dan mengembalikan string 24-karakter yang dapat dibaca seperti 'Tue Dec 11 18:07:14 2008'.
time.clock()	Mengembalikan waktu CPU saat ini sebagai jumlah floating-point detik. Untuk mengukur biaya komputasi dari berbagai pendekatan, nilai time.clock lebih bermanfaat daripada time.time ().
time.ctime([secs])	Seperti asctime (localtime (detik)) dan tanpa argumen seperti asctime ()
time.gmtime([secs])	Menerima instan yang diungkapkan dalam hitungan detik sejak zaman dan mengembalikan waktu tuple t dengan waktu UTC. Catatan: t.tm_isdst selalu 0
time.localtime([secs])	Menerima instan yang dinyatakan dalam hitungan detik sejak zaman dan mengembalikan waktu tuple t dengan waktu setempat (t.tm_isdst adalah 0 atau 1, tergantung pada apakah DST berlaku seketika oleh peraturan lokal).
time.mktime(tupletime)	Menerima instan dinyatakan sebagai time-tuple di waktu setempat dan mengembalikan nilai floating-

Fungsi Python	Penjelasan
	point dengan instan yang dinyatakan dalam hitungan detik sejak zaman.
time.sleep(secs)	Menangguhkan panggilan untuk beberapa detik.
time.strftime (fmt[,tupletime])	Menerima instan dinyatakan sebagai tupel waktu di waktu lokal dan mengembalikan sebuah string yang mewakili instan seperti yang ditentukan oleh string fmt.
time.strptime (str,fmt='%a %b %d %H:%M:%S %Y')	Parses str sesuai dengan format string fmt dan mengembalikan format instant-tuple.
time.time()	Mengembalikan waktu saat ini secara instan, jumlah detik mengambang beberapa detik sejak zaman itu.
time.tzset()	Mengatur ulang aturan konversi waktu yang digunakan oleh rutinitas perpustakaan. Variabel lingkungan TZ menentukan bagaimana hal ini dilakukan.

Modul waktu mempunyai dua atribut penting yaitu :

Method *Python*	Penjelasan
time.timezone	Atribut time.timezone adalah offset dalam detik zona waktu lokal (tanpa DST) dari UTC (> 0 di Amerika; <= 0 di sebagian besar Eropa, Asia, Afrika).
time.tzname	Atribut time.tzname adalah sepasang string yang bergantung pada lokal, yang merupakan nama zona waktu lokal tanpa dan dengan DST.

5. Modul calendar

Modul kalender berfungsi untuk menyimpan fungsi yang berkaitan dengan kalender, hal ini termasuk fungsi untuk mencetak kalender teks untuk bulan dan tahun tertentu.

Secara umum, kalender mengambil hari Senin sebagai hari pertama dalam satu minggu dan hari minggu sebagai hari ke tujuh atau terakhir. Untuk mengubah ini kita dapat menggunakan fungsi ***call calendar.setfirstweekday()***. Berikut adalah daftar fungsi-fungsi yang tersedia :

Fungsi Python	Penjelasan
calendar.calendar (year,w=2,l=1,c=6)	Mengembalikan string multiline dengan kalender untuk tahun tahun yang diformat menjadi tiga kolom yang dipisahkan oleh ruang c. W adalah lebar karakter setiap tanggal; Setiap baris memiliki panjang 21 * w + 18 + 2 * c. L adalah jumlah baris untuk setiap minggu.
calendar.firstweekday( )	Mengembalikan pengaturan saat ini untuk hari kerja yang dimulai setiap minggu. Secara default, saat kalender pertama kali diimpor, ini adalah 0, yang berarti Senin.
calendar.isleap(year)	Pengembalian True jika tahun adalah tahun kabisat; Jika tidak, False
calendar.leapdays(y1,y2)	Mengembalikan jumlah lompatan hari dalam tahun-tahun dalam rentang (y1, y2).
calendar.month (year,month,w=2,l=1)	Mengembalikan string multiline dengan kalender untuk bulan bulan tahun, satu baris per minggu ditambah dua baris header. W adalah lebar karakter setiap tanggal; Setiap baris memiliki panjang 7 * w + 6. L adalah jumlah baris untuk setiap minggu.
calendar.monthcalendar (year,month)	Mengembalikan daftar daftar int. Setiap sublist menunjukkan seminggu. Hari di luar bulan bulan tahun diatur ke 0; Hari dalam bulan ditetapkan ke hari ke bulan, 1 dan ke atas.
calendar.monthrange (year,month)	Mengembalikan dua bilangan bulat. Yang pertama adalah kode hari kerja untuk hari pertama bulan bulan di tahun; Yang kedua adalah jumlah hari dalam sebulan. Kode hari kerja adalah 0 (Senin) sampai 6 (Minggu); Angka bulan adalah 1 sampai 12.
calendar.prcal (year,w=2,l=1,c=6)	Seperti kalender cetak.calendar (tahun, w, l, c).
calendar.prmonth (year,month,w=2,l=1)	Seperti kalender cetak. Bulan (tahun, bulan, w, l).
calendar.setfirstweekday (weekday)	Mengatur hari pertama setiap minggu sampai hari kerja kode hari kerja. Kode hari kerja adalah 0 (Senin) sampai 6 (Minggu).
calendar.timegm (tupletime)	Kebalikan dari time.gmtime: menerima waktu instan dalam bentuk tupel waktu dan mengembalikan detik yang sama seperti jumlah floating-point dalam hitungan detik sejak zaman.
calendar.weekday (year,month,day)	Mengembalikan kode hari kerja untuk tanggal yang ditentukan. Kode hari kerja adalah 0 (Senin) sampai 6 (Minggu); Bulan adalah 1 (Januari) sampai 12 (Desember).

BAB 6 FUNGSI DAN MODUL

A. Fungsi

Fungsi adalah blok kode terorganisir yang dapat digunakan kembali untuk melakukan sebuah tindakan/*action*. Fungsi menghasilkan modularitas yang lebih baik untuk aplikasi kita dengan tingkat penggunaan kode yang lebih tinggi.

1. Mendefinisikan Fungsi

Berikut adalah aturan sederhana untuk mendefinisikan fungsi di dalam *Python*.

- Untuk membuat Fungsi dimulai dengan perintah **`def`** dan diikuti oleh nama fungsi dan tanda kurung **`(()).`**
- Setiap parameter masukan atau argumen harus ditaruh di dalam tanda kurung.
- Pernyataan pertama dari sebuah fungsi dapat berupa pernyataan opsional - string dokumentasi fungsi atau docstring.
- Blok kode dalam setiap fungsi diawali dengan tanda titik dua `(:)` dan indentasi.
- Pernyataan kembali `[ekspresi]` keluar dari sebuah fungsi, secara opsional menyampaikan kembali ekspresi ke pemanggil. Pernyataan pengembalian tanpa argumen sama dengan return None. Buatlah skrip berikut kemudian

```
#Contoh 1
def cetak_string(par1,par2):
    print ("NIM  Anda adalah = ",par1)
    print ("Nama Anda adalah = ",par2)


def hitung(a,b):
    print ("Hasil penjumlahan ",a,"+",b,"adalah",(a+b))

#main program
nama=input("Masukan NIm Anda = ")
nim= input("Masukan Nama Anda  = ")
cetak_string(nama,nim)
bil1=10
bil2=12
hitung(bil1,bil2)
```

Untuk contoh yang lebih jelas silahkan anda coba program berikut;

```
#Contoh2
def input_data():
    "fungsi pertama"
    nama= input("Masukan Nama:")
    nim= input("Masukan NIM:")

def cetak_string():
    print ("ini adalah fungsi cetak string")
    print ("silahkan masukan data")
    input_data()

cetak_string()
```

B. Modul

Modul berfungsi untuk memungkinkan kita mengatur kode *Python* secara logis, yaitu dengan mengelompokkan kode terkait ke dalam modul dan membuat kode lebih mudah dipahami dan digunakan. Modul adalah objek didalam *Python* dengan atribut yang diberi nama yang bisa kita bind dan dijadikan secara referensi.

Secara sederhana modul dapat kita pahami sebagai file yang terdiri dari kode *Python*, Modul dapat mendefinisikan fungsi, kelas

dan variabel. Modul juga bisa menyertakan kode yang bisa dijalankan secara *"runable"*.

Berikut adalah contoh modul sederhana pada *Python* : (simpan dengan nama fungsi1.py)

```
def cetak_fungsi( par ):
    print ("Selamat datang di FTIK USM : ", par)
    return
```

1. Import Statement

Setelah selesai membuat modul, maka untuk mengeksekusi modul tersebut agar dapat dijalankan secara eksternal (lewat program lain) maka kita dapat menggunakan perintah/sintak **`import`**, dengan menggunakan perintah **`import`** ini kita dapat menggunakan file sumber *Python* apapun sebagai modul dengan cara mengeksekusi pernyataan **`import`** tersebut di dalam file sumber *Python* lain-nya.

Ketika interpreter menemukan pernyataan import, ia akan mengimpor modul dan modul tersebut harus ada di jalur pencarian atau direktori yang ditentukan. Jalur pencarian ialah daftar direktori yang ditentukan sebelum mengimpor modul.

Misalnya, untuk mengimpor modul **fungsi1.py**, kita perlu meletakkan perintah berikut di bagian atas script.

```
# Import module fungsi1
import fungsi1

# kita bisa memanggil fungsi defined sebagai berikut
Fungsi1. cetak_fungsi ("Andy")
```

BAB 7 *LIST, TUPLE, DICTIONARY, OBJECK,* DAN *CLASS*

Di dalam bahasa pemrograman *Python*, struktur data yang paling dasar adalah lists atau urutan. Setiap elemen yang berurutan akan diberi nomor urut posisi atau sesuai dengan indeksnya. Indeks pertama dalam list adalah nol, indeks kedua adalah satu dan seterusnya.

Python mempunyai enam jenis urutan built-in, namun yang paling sering digunakan adalah list, tuple dan dictionary, akan tetapi ada beberapa hal yang dapat kita lakukan dengan semua jenis list. Operasi tersebut meliputi pengindeksan, pengiris, penambahan, perbanyak, dan pengecekan keanggotaan. Selain itu, *Python* memiliki fungsi built-in untuk menemukan panjang list dan untuk menemukan elemen terbesar dan terkecilnya.

A. *List*

1. Membuat *List*

List adalah tipe data yang paling serbaguna yang terdapat pada bahasa *Python*, yang dapat ditulis sebagai daftar nilai yang dipisahkan koma (item) antara tanda kurung siku. Hal penting tentang daftar adalah item dalam list tidak boleh sama jenisnya.

contoh sederhana pembuatan list dalam bahasa *Python*.

```
#Contoh sederhana pembuatan list
#pada bahasa pemrograman python
list1 = ['TI', 'SI', 'ILKOM', 2010, 2006]
list2 = [1, 2, 3, 4, 5 ]
list3 = ["x", "y", "z", "w"]
```

2. Akses Nilai dalam *List*

Untuk mengakses nilai yang terdapat di dalam list *Python*, kita dapat menggunakan tanda kurung siku untuk mengiris beserta indeks atau indeks agar mendapatkan nilai yang tersedia pada indeks tersebut.

```
#Cara mengakses nilai di dalam list Python

list1 = ['TI', 'SI', 'ILKOM', 2010, 2006]
list2 = ["x", "y", "z", "w"]

print ("list1[0]: ", list1[0])
print ("list2[1:5]: ", list2[1:5])
```

3. Update Nilai dalam *List*

Kita juga dapat memperbarui satu atau beberapa nilai di dalam list dengan cara memberikan potongan di sebelah kiri operator penugasan, dan kita juga dapat menambahkan nilai ke dalam list dengan *metode append ()*. Sebagai contoh berikut:

```
list1 = ['TI', 'SI', 'ILKOM', 2010, 2006]
print ("Nilai ada pada index 2 : ", list[2])

list[2] = 2001
print ("Nilai baru ada pada index 2 : ", list[2])
```

4. Hapus Nilai dalam *List*

Untuk menghapus nilai yang terdapat di dalam list *Python* juga mudah, kita cukup menggunakan perintah **del** jika kita tahu persis posisi elemen yang ingin kita hapus, atau kita dapat juga menggunakan **metode remove()** jika kita tidak tahu persis item mana yang akan dihapus. Sebagai contoh berikut:

```
#Contoh cara menghapus nilai pada list python
list1 = ['TI', 'SI', 'ILKOM', 2010, 2006]
print (list)
del list[2]
print ("Setelah dihapus nilai pada index 2 : ", list)
```

5. Operasi Dasar pada *List*

List Python juga dapat merespons operator **+** dan ***** seperti halnya **string**; Itu artinya operator penggabungan dan perulangan juga berlaku di sini, kecuali hasilnya adalah list baru, bukan sebuah String.

Sebenarnya, list merespons semua operasi urutan umum yang kita gunakan pada String di bab sebelumnya. Dibawah ini adalah tabel daftar operasi dasar pada list *Python*.

Python Expression	Hasil	Penjelasan
len([1, 2, 3, 4])	4	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Halo!'] * 4	['Halo!', 'Halo!', 'Halo!', 'Halo!']	Repetition
2 in [1, 2, 3]	TRUE	Membership
for x in [1,2,3] : print (x,end = ' ')	1 2 3	Iteration

6. *Indexing, Slicing* dan *Matrix* pada *List*

Karena list adalah urutan, pengindeksan dan pengiris bekerja dengan cara yang sama untuk list seperti yang mereka lakukan untuk String. Dengan asumsi input berikut :

```
L = ['C++'', 'Java', 'Python']
```

Python Expression	Hasil	Penjelasan
L[2]	'Python'	Offset mulai dari nol
L[-2]	'Java'	Negatif: hitung dari kanan
L[1:]	['Java', 'Python']	Slicing mengambil bagian

7. *Method* dan *Fungsi Build-in* pada *List*

Python Function	Penjelasan
cmp(list1, list2)	# Tidak lagi tersedia dengan Python 3
len(list)	Memberikan total panjang list.
max(list)	Mengembalikan item dari list dengan nilai maks.
min(list)	Mengembalikan item dari list dengan nilai min.
list(seq)	Mengubah tuple menjadi list.

Python Methods	Penjelasan
list.append(obj)	Menambahkan objek obj ke list
list.count(obj)	Jumlah pengembalian berapa kali obj terjadi dalam list
list.extend(seq)	Tambahkan isi seq ke list
list.index(obj)	Mengembalikan indeks terendah dalam list yang muncul obj
list.insert(index, obj)	Sisipkan objek obj ke dalam list di indeks offset
list.pop(obj = list[-1])	Menghapus dan mengembalikan objek atau obj terakhir dari list
list.remove(obj)	Removes object obj from list
list.reverse()	Membalik list objek di tempat
list.sort([func])	Urutkan objek list, gunakan compare func jika diberikan

B. *Tuple*

Tupel adalah sebuah urutan objek *Python* yang tidak dapat berubah, Tupel adalah urutan seperti daftar. Perbedaan utama antara Tupel dan List adalah bahwa data yang ada pada Tupel tidak dapat diubah, tidak seperti List *Python*. Tupel mengguna-kan symbol tanda kurung, sedangkan List *Python* menggunakan tanda kurung siku.

Membuat Tuple semudah memasukkan nilai-nilai yang dipisahkan koma. Secara opsional, kita dapat memasukkan nilai-nilai yang dipisahkan koma ini di antara tanda kurung juga. Sebagai contoh :

```
#Contoh sederhana pembuatan tuple
#pada bahasa pemrograman python

tup1 = ('fisika', 'kimia', 1993, 2017)
tup2 = (1, 2, 3, 4, 5 )
tup3 = "a", "b", "c", "d"
```

1. Akses Nilai dalam *Tuple*

Untuk mengakses nilai yang berada pada tupel, gunakan tanda kurung siku untuk mengiris beserta indeks atau indeks untuk mendapatkan nilai yang tersedia pada indeks tersebut. Sebagai contoh :

```
#Cara mengakses nilai tuple

tup1 = ('fisika', 'kimia', 1993, 2017)
tup2 = (1, 2, 3, 4, 5, 6, 7 )

print ("tup1[0]: ", tup1[0])
print ("tup2[1:5]: ", tup2[1:5])
```

2. Update Nilai dalam *Tuple*

Tupel tidak berubah, yang berarti kita tidak dapat memperbarui atau mengubah nilai elemen tupel. kita dapat mengambil bagian dari tupel yang ada untuk membuat tupel baru seperti ditunjukkan oleh contoh berikut.

```
tup1 = (20, 38.56)
tup2 = ('USM', 'Jaya')

# Aksi seperti dibawah ini tidak bisa dilakukan pada tuple python
# Karena memang nilai pada tuple python tidak bisa diubah
# tup1[0] = 100;

# Jadi, buatlah tuple baru sebagai berikut
tup3 = tup1 + tup2
print (tup3)
```

3. Hapus Nilai dalam *Tuple*

Menghapus elemen tuple individual tidak mungkin dilakukan. Tentu saja, tidak ada yang salah dengan meng-gabungkan tupel lain dengan unsur-unsur yang tidak diinginkan dibuang. Untuk secara eksplisit menghapus keseluruhan tuple, cukup gunakan del statement. Sebagai contoh

```
tup = ('USM', 'Jaya', 1988, 2019);

print (tup)
del tup;
print ("Setelah menghapus tuple : ")
print (tup)
```

4. Operasi Dasar pada *Tuple*

Seperti halnya List, Tupel juga merespons operator + dan * sama seperti **String**; berarti penggabungan dan perulangan di juga dapat berlaku juga sini, kecuali hasilnya adalah tupel baru, bukan string.

Sebenarnya, Tuple merespons semua operasi urutan umum yang kita gunakan pada String di bab sebelumnya. Dibawah ini adalah tabel daftar operasi dasar pada Tuple *Python*.

Python Expression	Hasil
len((1, 2, 3))	3
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)
('Halo!',) * 4	('Halo!', 'Halo!', 'Halo!', 'Halo!')
3 in (1, 2, 3)	TRUE
for x in (1,2,3) : print (x, end = ' ')	1 2 3

5. *Indexing, Slicing* dan *Matrix* pada *Tuple*

Karena Tupel adalah urutan, pengindeksan dan pengiris bekerja dengan cara yang sama untuk tupel seperti pada String, dengan asumsi masukan berikut. Dengan asumsi input berikut :

```
T = ('C++', 'Java', 'Python')
```

Python Expression	Hasil	Penjelasan
`T[2]`	'Python'	Offset mulai dari nol
`T[-2]`	'Java'	Negatif: hitung dari kanan
`T[1:]`	('Java', 'Python')	Slicing mengambil bagian

6. Fungsi *Build-in* pada *Tuple*

Python Function	Penjelasan
`cmp(tuple1, tuple2)`	# Tidak lagi tersedia dengan Python 3
`len(tuple)`	Memberikan total panjang tuple.
`max(tuple)`	Mengembalikan item dari tuple dengan nilai maks.
`min(tuple)`	Mengembalikan item dari tuple dengan nilai min.

C. *Dictonary*

Dictionary Python sendiri berbeda dengan List ataupun Tuple, hal ini dikarenakan setiap urutannya berisi key dan value. Setiap key dipisahkan dari value-nya dengan titik dua (:), item dipisahkan oleh koma, dan semuanya tertutup dalam kurung kurawal. *Dictionary* juga dapat kosong tanpa isi, caranya ditulis hanya dengan dua kurung kurawal, seperti ini: {}.

Nilai *dictionary* dapat berupa tipe apa pun, namun key harus berupa tipe data yang tidak berubah seperti string, angka, atau tupel.

1. Akses Nilai dalam *Dictionary*

Untuk mengakses elemen Dictionary, kita dapat menggunakan tanda kurung siku yang sudah dikenal bersama dengan key untuk mendapatkan nilainya. Berikut adalah contoh sederhananya :

```
#Contoh cara membuat Dictionary pada Python

dict = {'nama': 'udin', 'usia': 23, 'Class': 'Awal'}
print ("dict['nama']: ", dict['nama'])
print ("dict['Usia']: ", dict['usia'])
```

2. Update Nilai dalam *Dictionary*

Kita dapat memperbarui *Dictionary* dengan menambahkan entri baru atau pasangan nilai kunci, memodifikasi entri yang ada, atau menghapus entri yang ada seperti ditunjukkan pada contoh sederhana yang diberikan di bawah ini.

```
#Update dictionary python

dict = {'nama': 'udin', 'usia': 23, 'Class': 'Awal'}
dict['Usia'] = 25; # Mengubah entri yang sudah ada
dict['School'] = "USM Semarang" # Menambah entri baru

print ("dict['Usia']: ", dict['usia'])
print ("dict['School']: ", dict['School'])
```

3. Hapus Elemen *Dictionary*

Kita dapat menghapus elemen *Dictionary* individual atau menghapus keseluruhan isi *Dictionary*. kita juga dapat menghapus seluruh *Dictionary* dalam satu operasi.

Untuk menghapus seluruh *Dictionary* secara eksplisit, cukup gunakan del statement. Berikut adalah contoh sederhana :

```
#Contoh cara menghapus pada Dictionary Python

dict = {'nama': 'udin', 'usia': 23, 'Class': 'Awal'}

del dict['nama'] # hapus entri dengan key 'Nama'
dict.clear()     # hapus semua entri di dict
del dict         # hapus dictionary yang sudah ada

print ("dict['Usia']: ", dict['usia'])
print ("dict['School']: ", dict['School'])
```

4. Fungsi Build-in pada *Dictionary*

Fungsi Python	Penjelasan
`cmp(dict1, dict2)`	Membandingkan unsur keduanya.
`len(dict)`	Memberikan panjang total Dictionary. Ini sama dengan jumlah item dalam Dictionary.
`str(dict)`	Menghasilkan representasi string yang dapat dicetak dari Dictionary
`type(variable)`	Mengembalikan tipe variabel yang lulus. Jika variabel yang dilewatkan adalah Dictionary, maka akan mengembalikan tipe Dictionary.

5. Method Build-in pada *Dictionary*

Method Python	Penjelasan
`dict.clear()`	Menghapus semua elemen Dictionary
`dict.copy()`	Mengembalikan salinan Dictionary
`dict.fromkeys()`	Buat Dictionary baru dengan kunci dari seq dan nilai yang disetel ke nilai.
`dict.get(key, default=None)`	For key, nilai pengembalian atau default jika tombol tidak ada dalam Dictionary
`dict.has_key(key)`	Mengembalikan true jika key dalam Dictionary, false sebaliknya
`dict.items()`	Mengembalikan daftari dari pasangan tuple dictionary (key, value)
`dict.keys()`	Mengembalikan daftar key dictionary
`dict.setdefault(key, default=None)`	Mirip dengan get (), tapi akan mengatur dict [key] = default jika kunci belum ada di dict
`dict.update(dict2)`	Menambahkan pasangan kunci kata kunci dict2 ke dict
`dict.values()`	Mengembalikan daftar nilai dictionary

D. *Objeck* dan *Class*

Python sudah menjadi bahasa berorientasi objek sejak bahasa *Python* ini dibuat, untuk membuat dan menggunakan kelas dan objek pada *Python* sangat mudah. Pada materi ini kita akan membantu anda untuk memahami dalam penggunaan pemrograman berorientasi objek dengan *Python*.

Jika anda belum paham dengan pemrograman berorientasi objek (OOP), kita harus mempelajari terlebih dahulu agar kita dapat memahami konsep dasarnya.

1. Membuat *Class*

Sintak class digunakan untuk membuat kelas baru, untuk membuat class sintak **class** diikuti **nama kelas** dan **titik dua**.

```
class ClassNama:
     'Optional class documentation string'
     class_suite
```

```
#sample class.py
class Employee:
   'Common base class for all employees'
   empCount = 0

   def __init__(self, nama, salary):
      self.nama = nama
      self.salary = salary
      Employee.empCount += 1

   def displayCount(self):
     print "Total Employee %d" % Employee.empCount

   def displayEmployee(self):
      print "Nama : ", self.nama,  ", Salary: ", self.salary
```

2. Mengakses Atribut

Kita dapat mengakses atribut objek menggunakan dot operator dengan objek. Variabel kelas akan diakses dengan menggunakan nama kelas sebagai berikut :

```
emp1.displayEmployee()
emp2.displayEmployee()
print ("Total Employee %d" % Employee.empCount)
```

```
#Sample class2.py
class Employee:
   'Common base class for all employees'
   empCount = 0

   def __init__(self, nama, salary):
      self.nama = nama
      self.salary = salary
      Employee.empCount += 1

   def displayCount(self):
     print ("Total Employee %d" % Employee.empCount)

   def displayEmployee(self):
      print ("Nama : ", self.nama,  ", Salary: ", self.salary)
```

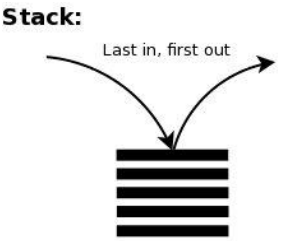
```
#This would create first object of Employee class"
emp1 = Employee("Udin", 2000)
#This would create second object of Employee class"
emp2 = Employee("Manni", 5000)
emp1.displayEmployee()
emp2.displayEmployee()
print ("Total Employee %d" % Employee.empCount)
```

BAB 8 *STACK* DAN *QUEUE*

A. Stack

Stack adalah tipe data yang bersifat *LIFO (Last In First Out)*, dimana data yang terakhir masuk akan menjadi data yang pertama kali dapat dikeluarkan. Struktur data stack mempunyai beberapa operasi antara lain adalah:

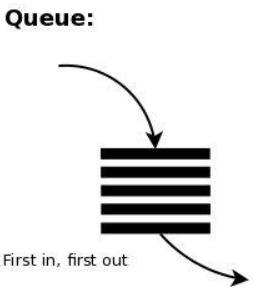
- `stack()`: Untuk membuat stack
- `push(item)`: Untuk menambahkan sebuah item ke dalam stack
- `pop()`: Untuk mengambil data dari stack (data paling atas akan dikeluarkan dari stack)
- `peek()`: Untuk mengidentifikasi data paling akhir yang masuk ke dalam stack
- `isEmpty()`: Untuk mengidentifikasi apakah stack masih kosong? Akan menghasilkan nilai TRUE jika stack masih dalam keadaan kosong
- `size()`: Untuk mengetahu jumlah item (elemen) dalam sebuah stack.



B. Queue

Struktur data *Queue* ini kebalikan dari *Stack* yaitu seperti sistem *antrian*, sistem queue menganut konsep *FIFO (First In First Out)*. Jenis operasi dalam *QUEUE*:

- `Queue()`: Membuat Queue baru
- `enqueue(item)` : Menambahkan data dalam queue, penambahan data selalu dilakukan dari bagian belakang (rear)
- `dequeue()`: Menghapus data, selalu dilakukan dari bagian depan (front)
- `isEmpty()`: Untuk mengetahui apakah Queue dalam keadaaan kosong atau tidak?
- `size()`: Untuk mengidentifikasi jumlah data yang ada didalam queue.




```
#contoh Stack.py
stack = ['USM','Jaya']; #inisialisasi stack

print ("stack awal :", stack)
stack.append("FTIK") #tambahkan elemen
stack.append("Joss") #tambahkan elemen
print("stack setelah ditambahkan : ", stack)

stack.pop() #hapus elemen terakhir
print("stack setelah elemen terakhir dihapus : ", stack)
```

```
#contoh deque.py
from collections import deque #import module

queue = deque(['USM','Jaya']); #inisialisasi queue

print ("queue awal :", queue)
queue.append("FTIK") #tambahkan elemen
queue.append("Joss") #tambahkan elemen
print("queue setelah ditambahkan : ", queue)

queue.popleft() #hapus elemen pertama
print("queue setelah elemen pertama dihapus : ", queue)
```

BAB 9 *SEARCHING*

Permasalahan yang sangat sering dihadapi dalam komputasi adalah *searching* (pencarian) dan *sorting* (pengurutan). *Searching* adalah algoritma untuk mencari item data yang terdapat di dalam sekumpulan item-item data yang lain. Hasil dari proses pencarian data ini berupa data boolean, yaitu betul atau satal, True atau False, tergantung pada apakah item data yang dicari ditemukan atau tidak ditemukan.

Di dalam *Python* terdapat sebuah operator yang dapat digunakan untuk pencarian data, yaitu operator `in` (keanggotaan), apakah sebuah data merupakan anggota dari sekolompok data. Contoh seperti pada kode berikut:

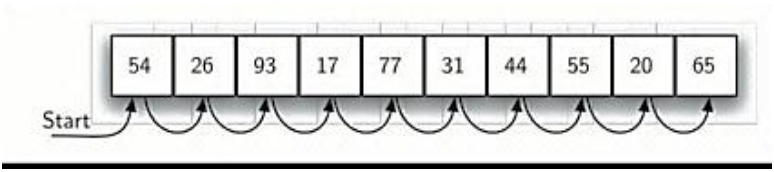
```
>>> 15 in [3,5,2,4,1]
False
>>> 3 in [3,5,2,4,1]
True
>>>
```

Sangat mudah untuk menggunakan operator in untuk mencari data, tetapi dalam bab ini kita akan mempelajari mengenai cara mengimplementasikan proses pencarian data.

A. *Sequential Search*

Ketika data disimpan dalam bentuk koleksi seperti *list* (*ordered*), maka dapat simpulkan bahwa data tersebut mempunyai hubungan yang bersifat *linear* atau *sequential*. Setiap data akan disimpan dengan posisi tertentu dan relatif terhadap data yang lain.

Di dalam *Python* posisi relatif tersebut merupakan *index* dari setiap item data, karena *index* dari sebuah item data adalah berupa urutan angka yang dimulai dari angka terkecil `(0,1,2,….)` maka sangat dimungkinkan untuk membaca setiap data dalam list secara berurutan (*sequential*), yang kemudian disebut dengan pencarian secara *sequential*. Seperti ilustrasi pada gambar di bawah ini:



Gambar di atas menjelaskan proses pencarian data dimulai dari item data pertama (ujung kiri, indeks = 0), kemudian dilanjutkan ke arah kanan secara berurutan sampai data yang dicari ditemukan, jika sampai posisi data yang terakhir maka data yang dicari tidak ditemukan maka hasilnya adalah False, atau data yang dicari tidak ditemukan.

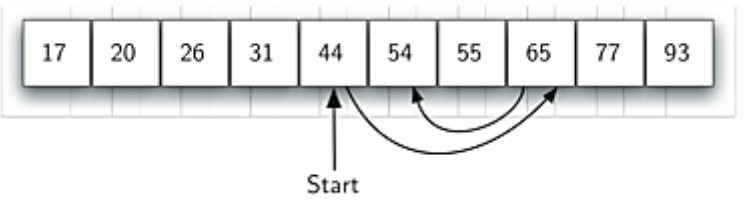
Berikut ini adalah contoh implementasi sequential searching dengan menggunakan Python.

```
def sequentialSearch(alist, item):
    pos=0
    found=False

    while pos < len (alist) and not found:
        if alist[pos]==item:
            found = True
        else :
            pos = pos+1

    return found

testlist=[1,2,32,8,17,19,42,13,0]
print ("Cari Nilai 3 :",sequentialSearch(testlist,5))
print ("Cari Nilai 42 :",sequentialSearch(testlist, 42))
```

B. *Binary Search*

Jika data kita adalah data yang telah diurutkan terlebih dahulu maka *Binary search* dapat melakukan pencarian lebih cepat jika dibanding dengan *sequential search*, dimana proses pencarian data dalam *binary search* akan dimulai dari bagian tengah *list*. Jika item yang dicari berada di tengah pencarian maka pencarian akan berakhir, tetapi jika item tidak ditemukan di tengah, pencarian berikutnya dapat dilakukan di setengah bagian atas atau setengah bagian bawah, hal ini tergantung pada besar atau kecilnya item data yang dicari. Proses ini akan diulang terus sampai data yang dicari ditemukan atau mencapai batas akhir *list/* item (tidak ditemukan).

Gambar di bawah adalah ilustrasi proses pencarain angka 54 dalam list, dengan menggunakan *binary search*.




```
def binarySearch(alist, item):
    first=0
    last=len(alist)-1
    found = False

    while first <= last and not found:
        midpoint = (first+last)//2
        if alist[midpoint]==item:
            found = True
        else :
            if item<alist[midpoint]:
                last = midpoint-1
            else :
                first=midpoint+1

    return found

testlist=[0,1,2,8,13,17,19,32,42]
print ("Cari Nilai 3 :",binarySearch(testlist,3))
print ("Cari Nilai 42 :",binarySearch(testlist, 42))
```

Algoritma *binary search* merupakan contoh yang sangat bagus dari strategi pencarian data yang disebut *divide and conquer*, yaitu dengan cara membagi kasus menjadi bagian yang lebih kecil.

BAB 10 *SORTING*

Sorting adalah proses menempatkan elemen data dalam sebuah koleksi data dengan urutan tertentu, sebagai contoh sebuah daftar kata-kata dapat diurutkan berdasarkan pada abjad (*alphabetically*) ataupun berdasarkan pada panjang pendeknya kata. Nama-nama mahasiswa dapat diurutkan berdasarkan pada nilai IPK, program studi ataupun asal kota.

Sampai sejauh ini sudah banyak algoritma sorting yang telah dikembangkan dan dianalisa. Hal ini sebagai bukti bahwa sorting adalah bagian penting dalam pembelajaran ilmu komputer *(computer science)*. *Sorting* item dalam jumlah yang sangat besar memerlukan *computing resource* yang besar pula, seperti halnya dengan algoritma *searching*, efisiensi algoritma *sorting* berkaitan erat dengan jumlah item yang diproses. Pada sebuah koleksi data dengan jumlah item sedikit, algoritma *sorting* yang komplek relative akan mengalamai masalah karena terlalu banyak persiapan data (*pre-prosessing*) yang harus dilakukan. Pada sisi lain, sebuah koleksi dengan jumlah item yang besar, diperlukan peningkatan strategi agar proses menjadi lebih cepat, meskipun strategi harus menggunakan strategi yang paling rumit sekalipun. Pada bagian ini akan dijelaskan beberapa teknik mengurutkan data dan membandingkannya berdasarkan pada waktu eksekusinya.

Sebelumnya, perlu kita perhatikan beberapa hal berkaitan dengan jenis operasi yang digunakan untuk menganalisa proses *sorting*. Pertama, perlu membandingkan dua nilai untuk melihat apakah nilainya lebih kecil atau lebih besar. Untuk mengurutkan

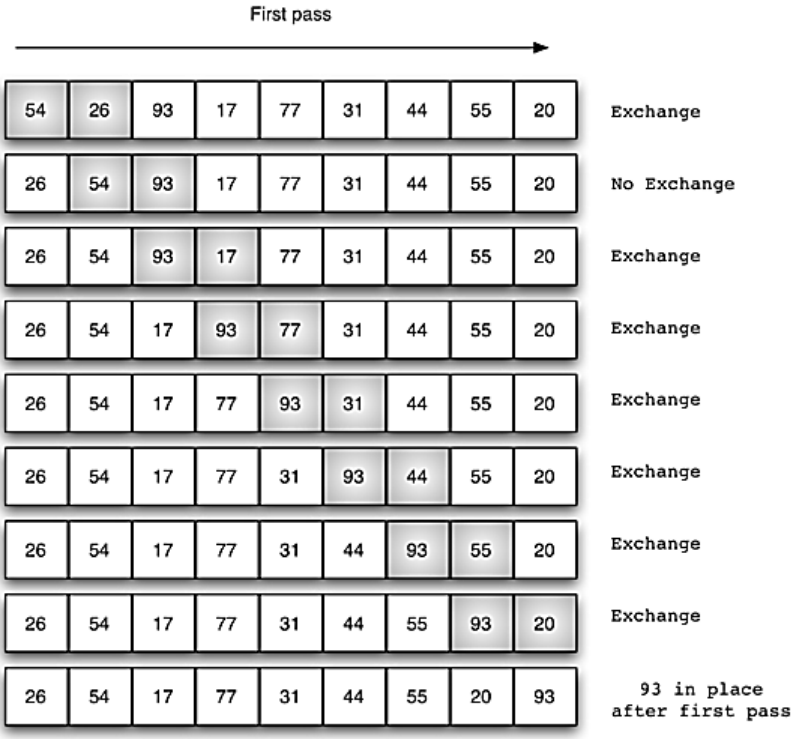
sebuah koleksi data, dibutuhkan beberapa cara yang sistematik untuk membandingkan nilainya untuk melihat apakah sebuah item keluar dari urutan. Jumlah operasi perbandingan umumnya merupakan cara yang paling banyak digunakan untuk mengukur proses pengurutan. Kedua, jika nilai sebuah item berada dalam posisi yang tidak benar, relatif terhadap posisi data yang lain, mungkin diperlukan untuk menukar posisi dari item tersebut. Menukar posisi sebuah item merupakan operasi yang memerlukan waktu dan *resource* dan jumlah operasi pertukaran seperti ini juga perlu dievaluasi untuk mendapatkan efisiensi keseluruhan dari sebuah algoritma.

A. *Bubble Sort*

Bubble sort mengurutkan data dengan cara memeriksa keseluruhan data secara berulang kali. *Bubble sort* membandingkan item-item yang bersebelahan dan akan menukar posisi dari item-item tersebut jika item tersebut belum urut. Setiap satu siklus pemeriksaan keseluruhan koleksi data akan diperoleh sebuah item paling besar pada posisi yang benar.

Silahkan anda perhatikan gambar di bawah untuk ilustrasi dari operasi operasi *bubble sort*. Item yang diberi tanda warna dasar abu-abu merupakan item-item yang sedang dibandingkan. Jika item sebelah kiri lebih besar maka posisinya akan ditukar, kondisi sebaliknya tidak perlu operasi penukaran tempat. Jika dalam koleksi data terdapat `n-item`, maka diperlukan perbandingan sebanyak `n-1`, pada siklus pertama. Perlu diperhatikan bahwa jika angka terbesar terdapat dalam daftar item yang sedang

dibandingkan, maka angka ini akan selalu dipindahkan ke samping kanan hingga ke posisi paling akhir.



Di awal siklus yang kedua, angka terbesar sudah berada pada posisi yang benar, yaitu di posisi paling kanan pada koleksi data. Selanjutnya masih terdapat n-1 item yang harus diurutkan, yang berarti masih ada n-2 pasang data yang harus dibandingkan, karena setiap siklus akan menempatkan item terbesar di posisi paling kanan, maka jumlah siklus keseluruhan proses adalah n-1 siklus. Setelah n-1 siklus pengurutan maka angka paling kecil akan berada di posisi paling kiri. Contoh kode di bawah adalah sebuah kode program yang merupakan implementasi dari algoritma *bubble sort*. Program ini akan memproses list dalam parameter dan melakukan

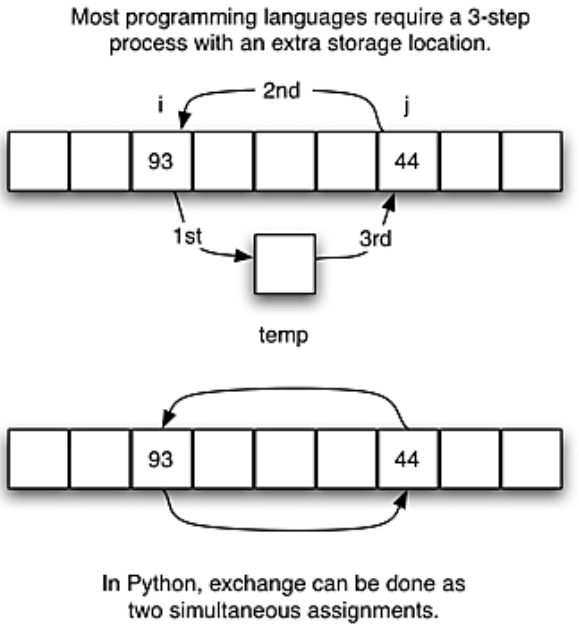
modifikasi dengan cara menukar posisi item data dalam list jika diperlukan.

```
def bubbleSort (alist):
    for passnum in range(len(alist)-1,0,-1):
        for i in range(passnum):
            if alist[i]>alist[i+1]:
                temp = alist[i]
                alist[i] = alist[i+1]
                alist[i+1] = temp

alist = [56,27,28,34,77,88,44,22,33,8]
bubbleSort(alist)
print(alist)
```

Operasi penukaran posisi item data terkadang disebut dengan operasi "*swap*". Biasanya operasi swap perlu sebuah memori tambahan yang bersifat sementara, seperti dalam potongan kode berikut:

```
temp = alist[i]
alist[i] = alist[j]
alist[j] = temp
```

Kode di atas akan menukar isi item i dan j dalam list. Di dalam *Python* sedikit berbeda, operasi penukaran data seperti di atas dapat dilakukan dengan operasi *assigment* secara simultan, contoh '**a, b = b, a**' perintah ini akan menukar isi data pada variabel a ke variabel b.

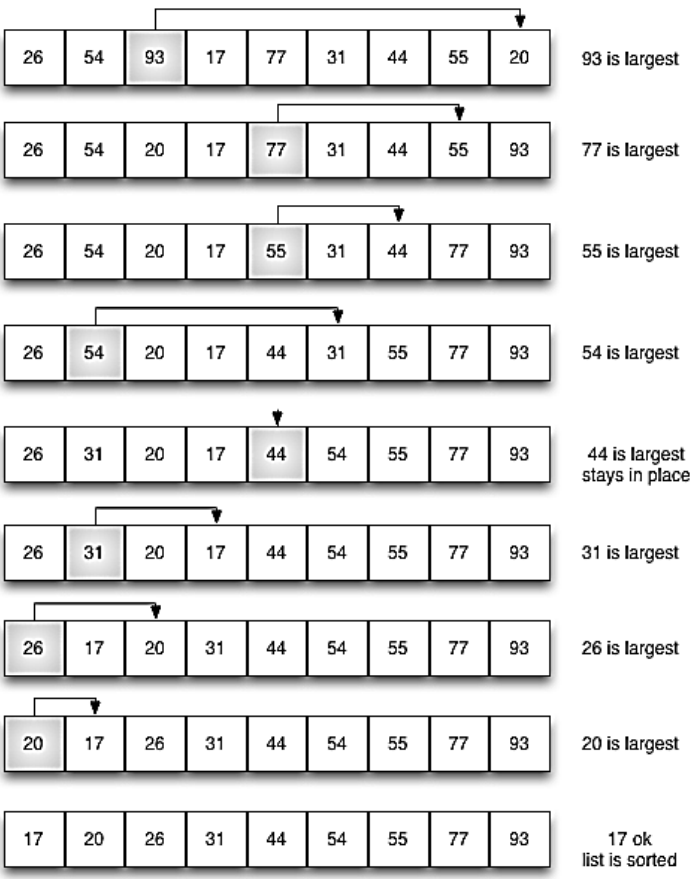


B. *Selection Sort*

Selection sort dapat memperbaiki performa *bubble sort* dengan hanya membuat satu pertukaran data saya untuk setiap siklusnya. Untuk melakukan hal ini, *selection sort* mencari lokasi dari angka terbesar untuk setiap satu siklus selanjutnya memindahkan (menukarkan) isinya di akhir siklus.

Setelah siklus pertama, angka terbesar berada di posisi paling kanan, seperti halnya pada *bubble sort*. Siklus kedua akan menempatkan angka terbesar berikutnya, dan proses ini akan berlanjut dan memerlukan n-1 siklus untuk mengurutkan item sebanyak n, item terakhir akan berada pada posisi yang benar setelah operasi siklus ke n-1.

Gambar di bawah menunjukkan ilustrasi bagaimana algoritma *selection sort* berjalan.



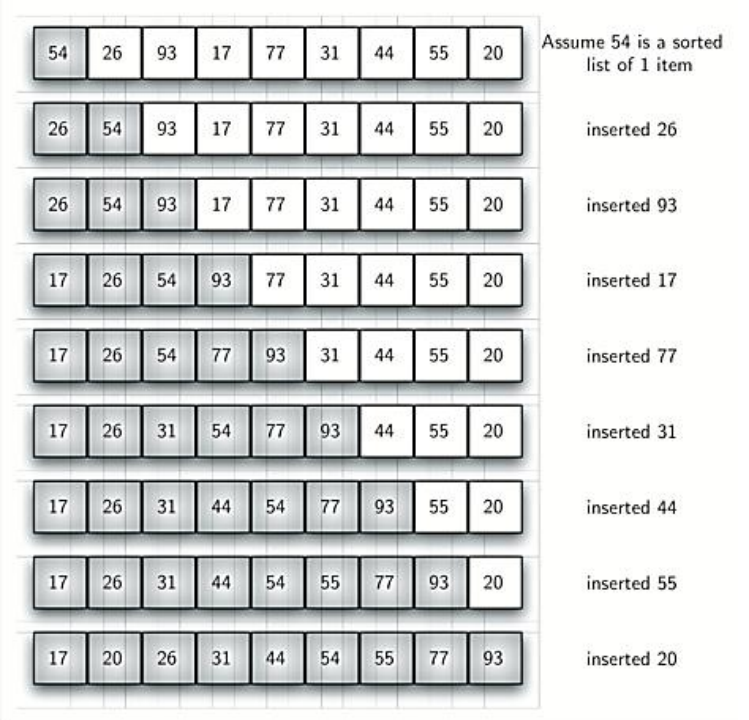
Untuk mengimplementasikan selection sort dapat di lihat kode program di bawah ini :

```
def selectionSort(alist):
   for fillslot in range(len(alist)-1,0,-1):
       positionOfMax=0
       for location in range(1,fillslot+1):
           if alist[location]>alist[positionOfMax]:
               positionOfMax = location

       temp = alist[fillslot]
       alist[fillslot] = alist[positionOfMax]
       alist[positionOfMax] = temp

alist = [54,26,93,17,77,31,44,55,20]
selectionSort(alist)
print(alist)
```

C. *Insertion Sort*

Meskipun *insertion sort* masih memiliki tingkat kompleksitas juga (dilihat dari jumlah operasi perbandingannya), namun demikian *insertion sort* bekerja dengan cara yang sedikit berbeda dengan algoritma *sorting* yang lainnya.

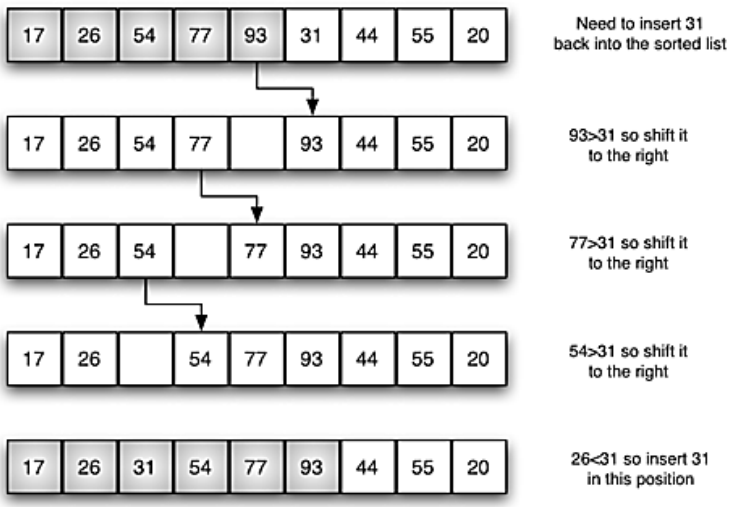
Insertion sort bekerja dengan cara selalu menjaga urutan data di bagian awal koleksi *(list)*. Tahap selanjutnya setiap item berikutnya akan di sisipkan pada bagian awal list pada posisi yang sudah urut. Gambar di bawah ini adalah ilustrasi proses pengurutan data dengan algoritma *insertion sort*. Item yang diberi latar belakang warna gelap menunjukan bagian awal list yang sudah terurut, untuk setiap siklusnya.



List dengan satu item pada posisi pertama (posisi 0) awalnya dianggap sudah dalam keadaan terurut. Pada setiap siklus berikutnya satu item data lain akan disisipkan pada posisi yang tepat di bagian awal list, sehingga akhirnya seluruh data dalam list akan urut dengan benar.

Pada contoh ilustrasi di atas pada siklus kelima, terdapat urutan data di bagian awal list terdiri atas angka 17, 26, 54, 77, dan 93. Selanjutnya akan disisipkan angka 31 ke dalam bagian awal list yang sudah urut. Operasi perbandingan dengan angka 93 menyebabkan terjadi operasi pergeseran angka 93 ke arah kanan, keadaan yang sama terjadi ketika melakukan operasi perbandingan dengan angka 77 dan 54. Ketika bertemu dengan item 26, operasi pergeseran posisi berhenti, dan terdapat 6 angka pada bagian awal

list dalam keadaan terurut. Seperti ditunjukkan pada gambar di bawah:



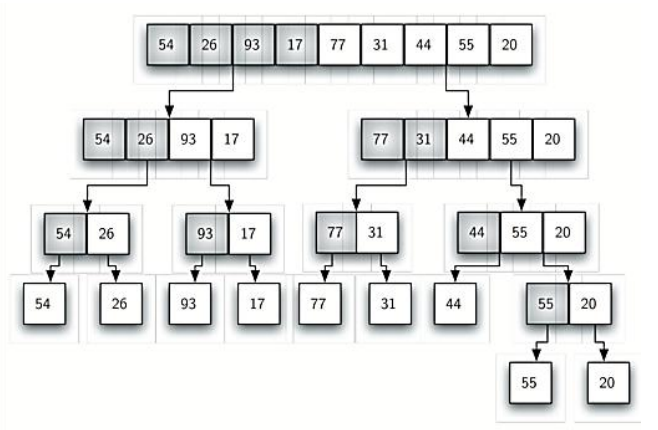
Kode program untuk implementasi algoritma *insertion sort* dapat dilihat pada gambar di bawah ini:

```
def insertionSort(alist):
   for index in range(1,len(alist)):

     currentvalue = alist[index]
     position = index

     while position>0 and alist[position-1]>currentvalue:
         alist[position]=alist[position-1]
         position = position-1

     alist[position]=currentvalue

alist = [54,26,93,17,77,31,44,55,20]
insertionSort(alist)
print(alist)
```

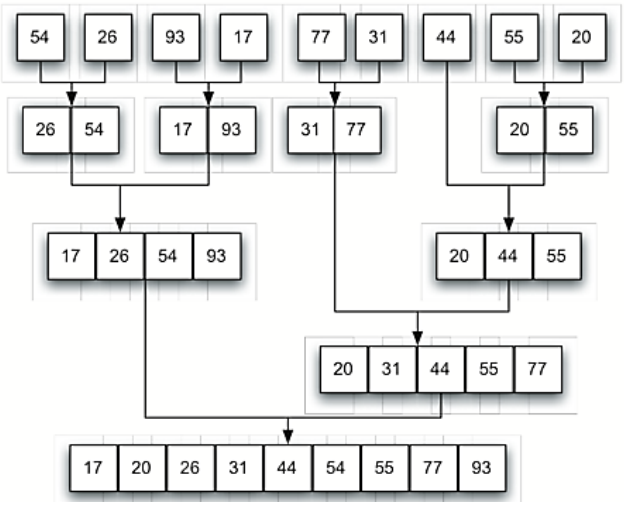
D. *Merge Sort*

Algoritma selanjunya adalah *merge sort*, algoritma ini untuk memperbaiki algoritma *sorting*, pada bagian ini dijelaskan strategi *sorting* dengan menggunakan **strategi devide and conquer.** Algoritma *sorting* dengan strategi *devide and conqure* yang pertama ialah *merge sort*. Algoritma *Merge sort* adalah algoritma rekursif yang selalu akan memecah list menjadi dua bagian. Jika list tidak berisi data (kosong) atau hanya ada satu item data maka dikatakan list tersebut dalam keadaan terurut. Jika list terdiri atas item data lebih dari satu, maka list akan dipecah menjadi dua *sublist* dan menerapkan *merge sort* pada kedua *sub-list*. Jika kedua *sublist* sudah terurut, operasi utama dalam *merge sort* diterapkan yaitu operasi penggabungan, atau merger. Yaitu operasi penggabungan dari dua *sublist* yang sudah terurut menjadi satu list yang baru yang terurut.

Proses merge sort diilustrasi pada gambar (a) tahapan devide and conqure dan (b) tahapan proses merge:



Gambar (a) merge sort



Gambar (b) merge sort

Kode program imlementasi algoritma *merge sort* ditunjuk-kan pada gambar di bawah ini:

```
def mergeSort(alist):
    print("Splitting ",alist)
    if len(alist)>1:
        mid = len(alist)//2
        lefthalf = alist[:mid]
        righthalf = alist[mid:]

        mergeSort(lefthalf)
        mergeSort(righthalf)

        i=0
        j=0
        k=0
        while i < len(lefthalf) and j < len(righthalf):
            if lefthalf[i] < righthalf[j]:
                alist[k]=lefthalf[i]
                i=i+1
            else:
                alist[k]=righthalf[j]
                j=j+1
            k=k+1
```

```
        while i < len(lefthalf):
            alist[k]=lefthalf[i]
            i=i+1
            k=k+1

        while j < len(righthalf):
            alist[k]=righthalf[j]
            j=j+1
            k=k+1
    print("Merging ",alist)

alist = [54,26,93,17,77,31,44,55,20]
mergeSort(alist)
print(alist)
```

Cara kerja program di atas adalah, sekali fungsi *merge sort* di panggil, maka fungsi akan memecah list menjadi dua *sublist* dan memanggil dirinya sendiri (rekursif, baris 8-9) untuk mengurutkan data pada masing-masing sub-list. Selanjutnya baris kode 11-31 bertanggung jawab untuk melakukan proses penggabungan (*merge*) dua *sublist* menjadi sebuah list yang terurut. Operasi penggabungan menempatkan setiap item data kembali ke list awal (*alist*) satu demi satu secara berulang dengan mengambil item paling kecil dari *sublist*.

Kode di atas dilengkapi dengan perintah *print* (baris-2) untuk menampilkan isi list yang sedang di urutkan pada awal setiap pemanggilan fungsi. Pada baris 32 juga terdapat perintah print untuk menampilkan proses penggabungan.

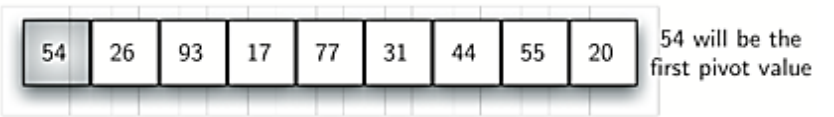
***** Catatan,

Pada saat list terdiri dari angka 44, 55, 20 maka proses memecahan list menjadi tidak berimbang. *Sublist* pertama terdiri atas 44, dan *sublist* kedua terdiri atas angka 55 dan 20.

E. *Quick Sort*

Algoritma *quick sort* juga menggunakan strategi *devide and conqure* seperti pada *merge sort*, tetapi tidak menggunakan memori tambahan, karena ada kemungkinan bahwa list tidak terbagi menjadi dua *sublist* yang sama jumlah anggotanya, hal ini dapat menjadi penyebab performansi menjadi menurun.

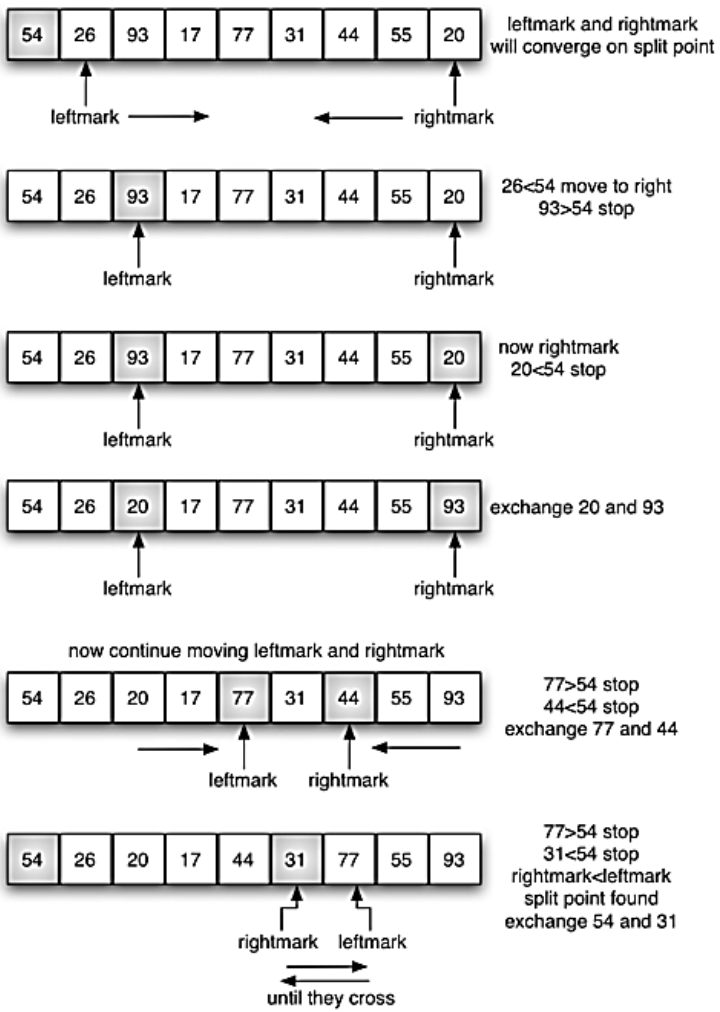
Pertama *quick sort* akan memilih sebuah nilai, yang disebut dengan *pivot value*, banyak cara yang dapat digunakan untuk memilih *pivot value*, dalam contoh ini *pivot value* yang digunakan adalah item pertama dalam list. Tugas dari *pivote value* ialah membantu memecah list menjadi dua *sublist*, posisi dimana *pivot value* berada disebut *split point*, selanjutnya akan berguna untuk memecah list untuk pemanggilan *quick-sort* berikutnya.



Gambar di atas menunjukkan bahwa angka 54 akan digunakan sebagai *pivot value*. Proses selanjutnya adalah memecah list menjadi dua *sublist*, yaitu dengan cara menemukan *split point* dan pada saat yang bersamaan memindahkan item-item lain ke sisi list, hal ini bergantung pada nilai item, jika lebih besar atau lebih kecil dibanding *pivot value*.

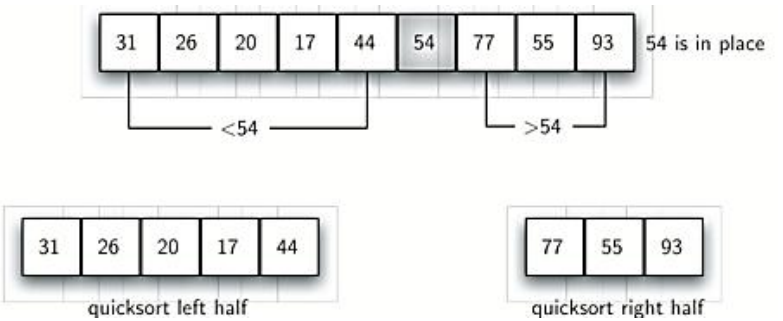
Proses partisi (memecah) list dimulai dengan menempatkan dua buah *marker*, kita sebut dengan istilah *leftmark* dan *rightmark*. *Leftmark* sesuai namanya akan ditempatkan pada sisi paling kiri list setelah *pivot value*, sedangkan *rightmark* di letakan pada posisi

paling kanan list (posisi 1 dan 8 pada gambar di bawah). Tujuannya adalah memindahkan item-item yang terletak di sisi yang salah yang dilihat dari sisi *pivot value*, selain itu juga mengabungkan item pada *split point*.



Proses selanjutnya ialah memeriksa item data dari posisi *leftmark* sampai ditemukan angka yang lebih besar dari *pivot value*.

Dari arah sebaliknya, *rightmark* dicari item yang nilainya lebih kecil dari *pivot value*. Dari proses ini, dari arah *leftmark* pencarian berhenti pada angka 93, sedangkan dari arah *rightmark* pencarian akan berhenti pada angka 20. Selanjutnya angka 93 dapat ditukar posisinya dengan angka 20, dan dilanjutkan proses yang sama sampai di peroleh kondisi dimana posisi *leftmask* > *rightmark*, dan pada saat kondisi itu terjadi maka posisi dari *rightmark* akan menjadi titik *split point*. Posisi *split point* sekarang dapat di tukar dengan posisi *pivot value* dan sekarang *pivot value* berada pada posisi yang benar, untuk lebih jelas dapat dilihat pada gambar di bawah:



Hasil dari pembagian list menjadi dua *sublist*, menyebabkan *sublist* di sebelah kiri *pivot value* memiliki elemen lebih kecil dari *pivot value* sedangkan *sublist* di sebelah kanan *pivot value* memiliki nilai lebih besar dari *pivot value*.

Selanjutnya pada setiap *sublist* diterapkan *quicksort* yang sama secara rekursif. Implementasi algoritma *quicksort* dapat dilihat pada gamabr berikut:

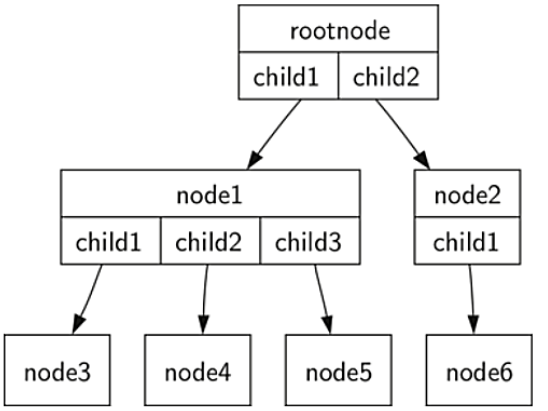
```
def quickSort(alist):
   quickSortHelper(alist,0,len(alist)-1)

def quickSortHelper(alist,first,last):
   if first<last:

       splitpoint = partition(alist,first,last)

       quickSortHelper(alist,first,splitpoint-1)
       quickSortHelper(alist,splitpoint+1,last)


def partition(alist,first,last):
   pivotvalue = alist[first]

   leftmark = first+1
   rightmark = last

   done = False
   while not done:

       while leftmark <= rightmark and alist[leftmark] <= pivotvalue:
           leftmark = leftmark + 1

       while alist[rightmark] >= pivotvalue and rightmark >= leftmark:
           rightmark = rightmark -1

       if rightmark < leftmark:
           done = True
       else:
           temp = alist[leftmark]
           alist[leftmark] = alist[rightmark]
           alist[rightmark] = temp

   temp = alist[first]
   alist[first] = alist[rightmark]
   alist[rightmark] = temp


   return rightmark

alist = [54,26,93,17,77,31,44,55,20]
quickSort(alist)
print(alist)
```

BAB 11 *TREE* DAN ALGORITMANYA

A. Definisi *Tree*

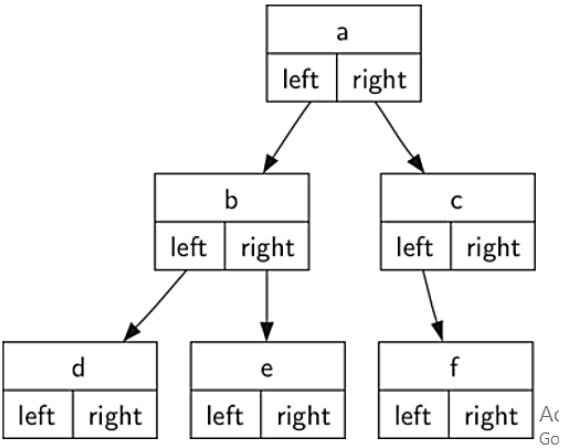
Sebuah *Tree* terdiri atas sekelompok *node* dan *edge* yang menghubungkan sepasang *node*. Tree mempunyai sifat-sifat sebagai berikut:

- Salah satu *node* dalam tree berfungsi sebagai *root*.
- Setiap *node* ***n***, kecuali *node root*, dihubungkan oleh *edge* dari *node* **p** yang lain dimana *node* ***p*** adalah *parent* dari *node* ***n***, dengan kata lain setiap *node* kecuali *root* pasti punya parent.
- Terdapat PATH yang bersifat unik dari *root* menuju ke setiap *node*.
- Jika setiap *node* dalam *tree* memiliki maksimum 2 *children* (*node*), maka *tree* tersebut disebut sebagai sebuah ***binary tree***.



B. Implementasi *Binary Tree* dengan menggunakan *Class*

Contoh struktur data *binary tree*



Dekalrasi kelas Tree:

class BinaryTree:

```
  # inisialisasi Root node
  def __init__(self,rootObj):
          self.key = rootObj
          self.leftChild = None  # belum ada children
          self.rightChild = None

  def insertLeft(self,newNode):
      if self.leftChild == None:
          self.leftChild = BinaryTree(newNode)
      else:
          t = BinaryTree(newNode)
          t.leftChild = self.leftChild
          self.leftChild = t

  def insertRight(self,newNode):
      if self.rightChild == None:
          self.rightChild = BinaryTree(newNode)
      else:
          t = BinaryTree(newNode)
          t.rightChild = self.rightChild
          self.rightChild = t
```

```
  # membaca children jalur kanan
  def getRightChild(self):
      return self.rightChild
  # membaca children dari jalur kiri
  def getLeftChild(self):
      return self.leftChild
  # memberi label Root
  def setRootVal(self,obj):
      self.key = obj
  # membaca label Root
  def getRootVal(self):
      return self.key
```

C. *Tree Traversal*

Cara menelusuri setiap *node* di dalam *Tree*

Preorder:

Penelusuran *node* berawal dari *root,* kemudian berlanjut ke turunan dari sebelah kiri, jika jalur kiri sudah tidak ada anak lagi, diteruskan ke jalur turunan sebelah kanan, terus sampai akhir node. Atau pertama menuju ke *node root* kemudian secara recursive melakukan *preorder* pada *subtree* sebelah kiri, dan diikuti dengan traversal *preorder* secara recursive ke *subtree* sebelah kanan.

Implementasi kode:

```
def preorder(tree):
    if tree:
        print(tree.getRootVal())
        preorder(tree.getLeftChild())
        preorder(tree.getRightChild())
```

Inorder:

Melakukan *traversal* inorder secara *recursive* pada *subtree* sebelah kiri, menuju ke *node root*, dan yang terakhir melakukan *traversal inorder* secara *recursive* pada *subtree* yang berada pada sebelah kanan.

Implementasi Kode:

```
def inorder(tree):
  if tree != None:
      inorder(tree.getLeftChild())
      print(tree.getRootVal())
      inorder(tree.getRightChild())
```

Postorder:

Melakukan *traversal postorder* secara *recursive* pada *subtree* bagian kiri dan kanan diikuti dengan berkunjung ke *node root*.

```
def postorder(tree):
    if tree != None:
        postorder(tree.getLeftChild())
        postorder(tree.getRightChild())
        print(tree.getRootVal())
```

D. Istilah dalam *Tree*

Node adalah serupa dengan simpul, *node* merupakan kerangka dasar dari sebuah *tree* (pohon), terkadang disebut sebagai "*key*". Sebuah *node* dapat berisi informasi tambahan (atau *payload*), meskipun informasi dalam *node* ini tidak wajib tetapi sangat penting dalam implementasi struktur data *tree*.

Edge adalah garis penghubung antara dua buah *node* yang menunjukkan adanya hubungan (relasi) diantara kedua *node*. Setiap

node (kecuali *node Root*) terhubung dengan satu penguhubung yang menuju ke dalam *node* dan beberapa penghubung yang mengarah keluar dari pada node.

Root adalah satu-satunya *node* dalam tree yang tidak memiliki garis penghubung (*edge*) yang mengarah ke dalam *Root* atau disebut dengan *parent*, *edge* yang menghubungkan *Root* semua mengarah ke luar dari node. (cikal-bakal)

Path adalah jalur yang menghubungkan antara satu *node* dengan *node* yang lain, dapat terdiri dari beberapa *node* dan *edge* yang berkesinambungan.

Children adalah *Node* yang terhubung dengan *edge* yang mengarah ke dalam *node* disebut children (selalu memiliki parent)

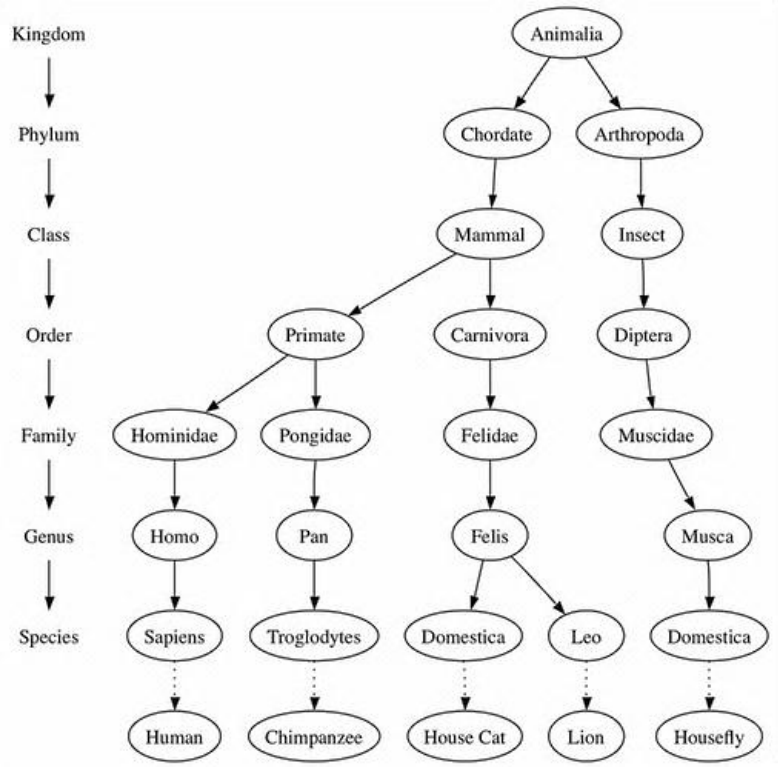
Parent adalah *Node* yang terhubung dengan *edge* yang mengarah keluar (memiliki anak).

Sibling adalah Node-*node* yang memiliki parent yang sama.

Subtree adalah sekelompok *node* beserta turunannya (anak-anaknya).

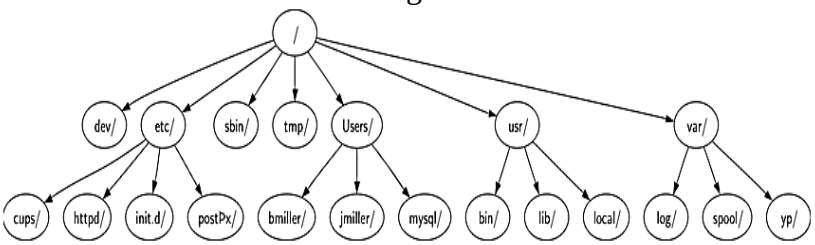
Leaf node adalah sebuah *node* yang tidak memiliki *children* (anak)

Level:



Felis berada pada tree **level 5**

Height:



Sebuah *tree* dengan height = 2

DAFTAR PUSTAKA

Belajar*Python*. (n.d.). Retrieved from https://belajar*Python*.com/

Duniailkom. (2019). Retrieved from Duniailkom: https://www.duniailkom.com/tutorial-belajar-bahasa-pemrograman-*Python*-untuk-pemula/

envantotuts+. (n.d.). Retrieved from https://code.tutsplus.com/categories/*Python*

Indonesia, P. (n.d.). Retrieved from https://www.*Python*indo.com/

Python. (n.d.). Retrieved from https://www.*Python*.org/doc/

sakti. (n.d.). Retrieved from https://sakti.github.io: https://sakti.github.io/*Python*101/struktur_data.html

Raharjo, B. (2017). Belajar Singkat Pemrograman *Python* 3. Bandung: Modula.

W3Schools. (n.d.). Retrieved from https://www.w3schools.com/*Python*/