

Hanya dipergunakan di lingkungan Fakultas Teknik Elektro



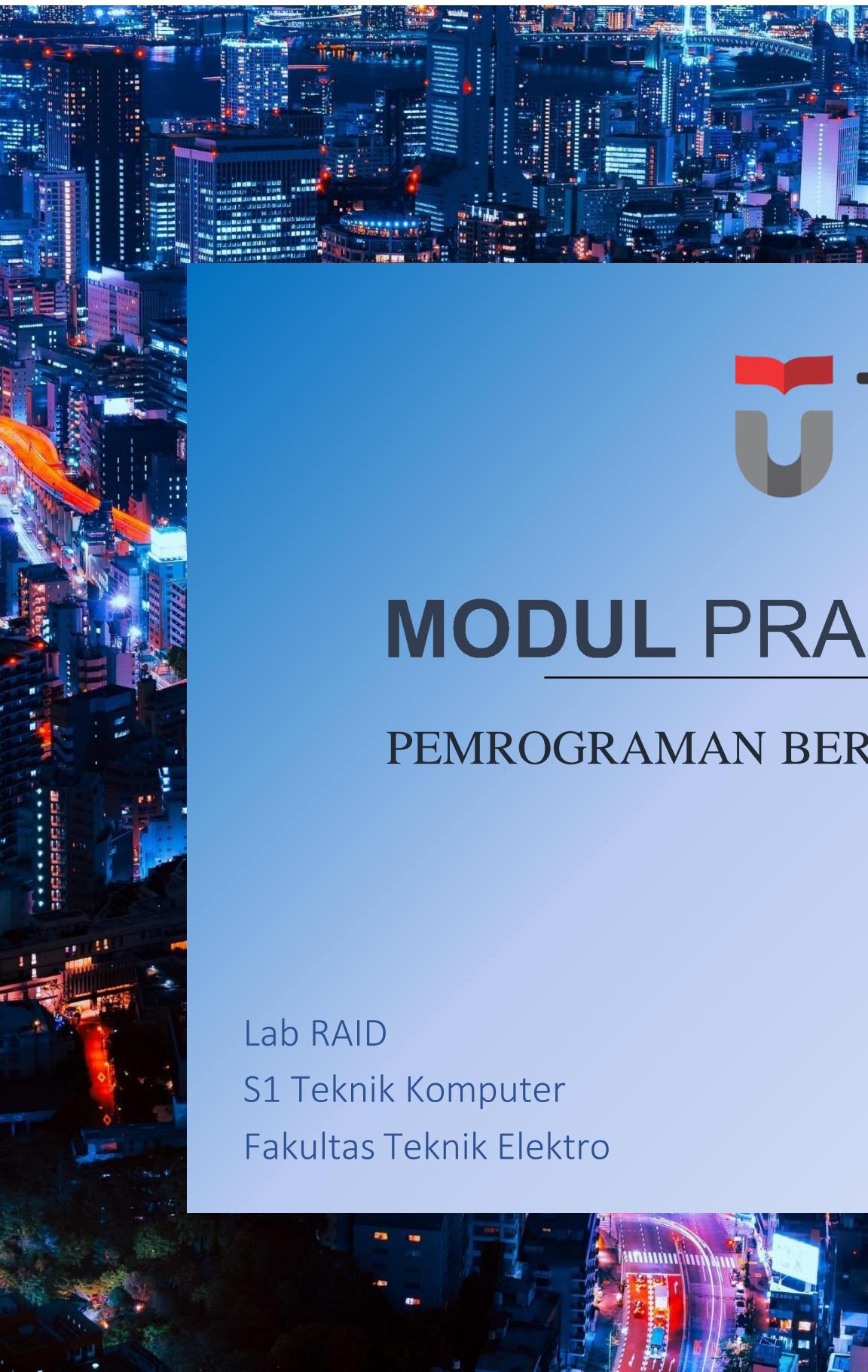
MODUL PRAKTIKUM

PEMROGRAMAN BERBASIS OBJEK

Lab RAID

S1 Teknik Komputer

Fakultas Teknik Elektro



DAFTAR PENYUSUN

- Casi Setianingsih, S.T., M.T
- Burhanuddin Dirgantoro, Ir. MT
- Ashri Dinimaharawati, ST. MT
- Asisten Laboratorium RAID

Lembar Revisi

Yang bertanda tangan di bawah ini:

Nama : Casi Setianingsih, S.T., M.T.

NIP 19890019

Jabatan : Dosen Pembina Laboratorium Realm of Artificial Intelligence
and Design

Dengan ini menyatakan pelaksanaan Revisi Modul Pemrograman Berbasis Objek untuk Prodi S1 Teknik Komputer, telah dilaksanakan sebagai berikut:

No	Keterangan Revisi	Tanggal Revisi Terakhir

LEMBAR PERNYATAAN

Yang bertanda dibawah ini:

Nama : Casi Setianingsih, S.T., M.T.

NIP 19890019

Jabatan : Dosen Pembina Laboratorium Realm of Artificial Intelligence
and Design

Menerangkan dengan sesungguhnya bahwa modul praktikum ini telah direview dan akan digunakan untuk pelaksanaan praktikum di Semester Ganjil Tahun Akademik 2022/2023 di Laboratorium Software Engineering & Application Fakultas Teknik Elektro Universitas Telkom.

Bandung, 26 July 2023

Mengetahui,

Ketua Kelompok Keahlian



Dr. Yudha Purwanto, S.T., M.T

NIP. 02770066

Dosen Pembina Laboratorium Realm
of Artificial Intelligence and Design



Casi Setianingsih, S.T., M.T

NIP. 19890019

STRUKTUR ORGANISASI

LABORATORIUM SOFTWARE ENGINEERING AND APPLICATION

TAHUN AJARAN 2022/2023

Dosen Pembina Lab:

Casi Setianingsih, S.T., M.T NIP. 19890019

Koordinator Asisten Praktikum:

Faraday Barr Fatahillah 1103213028

Asisten Praktikum:

Cetta Maulana Andhika	1103213119
Aldo Nitehe Lase	1103213110
Kinanti Rahayi Az-Zahra	1103210052
Sam Alim Ramadhan	1103210236
Azmi Taqiuddin Syah	1103213078
Muhammad Farrel Ahadi Tama	1103210177
Andreas Wahyu Prayogo	1103213114
Fadhil Dzikri Aqila	1103213136
Axel David	1103210017
Sulthan Nauval Abdillah	1103213065
Kinanti Aria Widaswara	1103213165
Muhammad Abyan Ridhan Siregar	1103210053
Yohanes Yogas Herlambang	1103210201

Panji Christoper Silalahi 1103213059

Sef Sofa Maulanaja 1103213095

Visi & Misi

Fakultas Teknik Elektro

VISI:

Menjadi fakultas berstandar internasional yang berperan aktif dalam pengembangan pendidikan, riset, dan entrepreneurship di bidang teknik elektro dan teknik fisika, berbasis teknologi informasi.

MISI:

1. Menyelenggarakan sistem pendidikan yang berstandar internasional di bidang teknik elektro dan teknik fisika berbasis teknologi informasi.
2. Menyelenggarakan, menyebarluaskan, dan memanfaatkan hasil-hasil riset berstandar internasional di bidang teknik elektro dan fisika.
3. Menyelenggarakan program entrepreneurship berbasis teknologi bidang teknik elektro dan teknik fisika di kalangan sivitas akademika untuk mendukung pembangunan ekonomi nasional.
4. Mengembangkan jejaring dengan perguruan tinggi dan industri terkemuka dalam dan luar negeri dalam rangka kerjasama pendidikan, riset, dan entrepreneurship.
5. Mengembangkan sumberdaya untuk mencapai keunggulan dalam Pendidikan, riset, dan entrepreneurship.

Visi & Misi

Jurusan Teknik Komputer

VISI:

“Menjadi Program Studi S1 Teknik Komputer berstandar internasional yang menghasilkan lulusan di bidang komputer.”

MISI:

1. Menyelenggarakan Pendidikan yang berstandar internasional untuk menghasilkan lulusan yang menguasai ilmu dan teknologi komputer.
2. Menyelenggarakan penelitian berkualitas internasional di bidang Sistem Komputer berbasis teknologi ilmu dan teknologi komputer dengan melibatkan mahasiswa secara aktif.
3. Menjalankan pengabdian masyarakat dengan prinsip menyebarluaskan ilmu dan teknologi komputer hasil penelitian kepada masyarakat luas dengan melibatkan mahasiswa secara aktif.
4. Membekali mahasiswa ilmu dan pengetahuan yang praktis, agar mampu bekerja, dan mengembangkan diri dan berwirausaha di bidang teknologi informasi dan komunikasi komputer.



ATURAN LABORATORIUM FAKULTAS TEKNIK ELEKTRO TELKOM UNIVERSITY

Setiap Mahasiswa Fakultas Teknik Elektro yang akan menggunakan Fasilitas Laboratorium, **WAJIB**

Mematuhi Aturan sebagai berikut :

1. Menggunakan seragam resmi Telkom University, dan membawa Kartu Tanda Mahasiswa (KTM) yang masih berlaku.
2. Rambut rapih.
3. Dilarang **merokok** dan **makan minum** didalam ruangan, dan membuang sampah ada tempatnya.
4. Dilarang menyimpan barang-barang milik pribadi di Laboratorium tanpa seijin Fakultas.
5. Dilarang menginap di Laboratorium tanpa seijin Fakultas.
6. Jam Kerja Laboratorium dan Ruang Riset adalah 06.30 WIB sampai 22.00 WIB.
7. Mahasiswa yang akan menggunakan Laboratorium dan atau ruang riset diluar jam kerja, harus mengajukan ijin kepada Fakultas.

Bandung, 26 July 2023

Dekan Fakultas Teknik Elektro

Dr. Bambang Setia Nugroho, S.T., M.T

DAFTAR ISI

Lembar Revisi.....	3
LEMBAR PERNYATAAN	4
STRUKTUR ORGANISASI	5
ATURAN LABORATORIUM FAKULTAS TEKNIK ELEKTRO	9
DAFTAR ISI.....	10
DAFTAR GAMBAR	12
DAFTAR TABEL.....	17
Modul 1: Pengenalan dan Dasar Python.....	19
1.1 Tujuan Praktikum	19
1.2 Alat dan Bahan:	19
1.3 Dasar Teori	19
1.3.1 Sejarah Python	19
1.3.2 Aplikasi dan Penggunaan Python.....	19
1.3.3 Konsep Dasar Pemrograman Python	20
Modul 2: Struktur Data	45
2.1 Tujuan Praktikum	45
2.2 Alat dan Bahan	45
2.3 Dasar Teori	45
2.3.1 List	45
2.3.2 Tuple	49
2.3.3 Set.....	55
2.3.4 Dictionary.....	58
Modul 3: Fungsi, Class dan Inheritance.....	65
3.1 Tujuan Praktikum	66
3.2 Alat dan Bahan	66
3.3 Dasar Teori	66
3.3.1 Fungsi	66
3.3.2 Class	71
3.3.3 Inheritance.....	74
Modul 4: Polymorphism & Encapsulation.....	85
4.1 Tujuan Praktikum	85
4.2 Alat dan Bahan	85
4.3 Dasar Teori	85

4.3.1	Polymorphism	85
4.3.2	Kelas Abstrak	88
4.3.3	<i>Encapsulation</i> (Enkapsulasi)	89
Modul 5:	Koneksi Database MySQL	91
5.1	Tujuan Praktikum	91
5.2	Alat dan Bahan	91
5.3	Dasar Teori	91
5.3.1	Database MySQL	91
5.3.2	Menghubungkan Python dengan MySQL.....	91
5.3.3	CRUD (Create, Read, Update, Delete)	94
Modul 6:	Web Scraping	103
6.1	Tujuan Praktikum	103
6.2	Alat dan Bahan	103
6.3	Dasar Teori	103
6.3.1	Konsep Web Scraping.....	103
6.3.2	Library Requests dan Beautiful Soup.....	103
6.3.3	Mengolah Data	108
6.3.4	GUI Automation dengan pywinauto	110
Daftar Pustaka.....		111

DAFTAR GAMBAR

Gambar 1.1 <i>Source Code Print Statement</i>	20
Gambar 1.2 <i>Source Code Print Statement</i>	20
Gambar 1.3 <i>Source Code Tipe Data Dasar - Numbers</i>	21
Gambar 1.4 <i>Source Code Tipe Data Dasar - Numbers</i>	21
Gambar 1.5 <i>Source Code Tipe Data Dasar - String</i>	22
Gambar 1.6 <i>Output Tipe Data Dasar - String</i>	22
Gambar 1.7 <i>Source Code Tipe Data Dasar – String Assignment</i>	23
Gambar 1.8 <i>Output Tipe Data Dasar – String Assignment</i>	23
Gambar 1.9 <i>Source Code Tipe Data Dasar – String Slicing</i>	23
Gambar 1.10 <i>Output Print Statement</i>	23
Gambar 1.11 <i>Source Code Tipe Data Dasar – String Concatenation</i>	24
Gambar 1.12 <i>Output Tipe Data Dasar – String Concatenation</i>	24
Gambar 1.13 <i>Source Code Operator Aritmetika – Penjumlahan dan Pengurangan</i>	25
Gambar 1.14 <i>Output Operator Aritmetika – Penjumlahan dan Pengurangan</i>	25
Gambar 1.15 <i>Source Code Operator Aritmetika – Perkalian dan Pembagian</i>	25
Gambar 1.16 <i>Output Operator Aritmetika – Perkalian dan Pembagian</i>	26
Gambar 1.17 <i>Source Code Operator Aritmetika – Floored Division</i>	26
Gambar 1.18 <i>Output Operator Aritmetika – Floored Division</i>	26
Gambar 1.19 <i>Source Code Operator Aritmetika – Modul</i>	27
Gambar 1.20 <i>Output Operator Aritmetika – Modulo</i>	27
Gambar 1.21 <i>Source Code Operator Perbandingan</i>	28
Gambar 1.22 <i>Output Operator Perbandingan</i>	28
Gambar 1.23 <i>Source Code Input</i>	29
Gambar 1.24 <i>Input – menunggu input user</i>	29
Gambar 1.25 <i>Input – Hasil</i>	29
Gambar 1.26 <i>Input – Failed</i>	29
Gambar 1.27 <i>Source Code Input – Multiplikasi</i>	30
Gambar 1.28 <i>Input – Multiplikasi</i>	30
Gambar 1.29 <i>Source Code Import Module</i>	31
Gambar 1.30 <i>Output Import Module</i>	31
Gambar 1.31 <i>Source Code Import from Module</i>	31
Gambar 1.32 <i>Output Import from Module</i>	31
Gambar 1.33 <i>Source Code Import As</i>	32
Gambar 1.34 <i>Output Import As</i>	32
Gambar 1.35 <i>Source Code Struktur Kondisional</i>	34
Gambar 1.36 <i>Output Struktur Kondisional if-true</i>	34
Gambar 1.37 <i>Output Struktur Kondisional if-false</i>	34
Gambar 1.38 <i>Source Code Struktur Kondisional if-else</i>	35
Gambar 1.39 <i>Output Struktur Kondisional if-else (true)</i>	35
Gambar 1.40 <i>Output Struktur Kondisional if-else (false)</i>	35
Gambar 1.41 <i>Source Code Struktur Kondisional if-elif-else</i>	36
Gambar 1.42 <i>Output if-elif-else (if true)</i>	36
Gambar 1.43 <i>Output if-elif-else (elif true)</i>	36
Gambar 1.44 <i>if-elif-else (else true)</i>	36

Gambar 1.45 <i>Source Code</i> Perulangan <i>for</i>	37
Gambar 1.46 <i>Source Code</i> Perulangan <i>for</i>	37
Gambar 1.47 <i>Output</i> Perulangan <i>for</i>	37
Gambar 1.48 <i>Output</i> Perulangan <i>for</i>	37
Gambar 1.49 <i>Source Code</i> Perulangan <i>While</i>	38
Gambar 1.50 <i>Output</i> Perulangan <i>While</i>	38
Gambar 1.51 <i>Source Code</i> Perulangan Bertingkat.....	39
Gambar 1.52 <i>Output</i> Perulangan Bertingkat.....	39
Gambar 1.53 <i>Source Code</i> Kontrol Perulangan <i>Break</i>	40
Gambar 1.54 <i>Output</i> Kontrol Perulangan <i>Break</i>	40
Gambar 1.55 <i>Source Code</i> Kontrol Perulangan <i>Continue</i>	41
Gambar 1.56 <i>Output</i> Kontrol Perulangan <i>Continue</i>	41
Gambar 1.57 <i>Source Code</i> Kontrol Perulangan <i>Pass</i>	42
Gambar 1.58 <i>Output</i> Kontrol Perulangan <i>Pass</i>	42
Gambar 1.59 <i>Source Code</i> Kontrol Perulangan <i>Pass</i>	43
Gambar 1.60 <i>Output</i> Kontrol Perulangan <i>While-Else</i>	43
Gambar 1.61 <i>Source Code</i> Kontrol Perulangan <i>For-Else</i>	44
Gambar 1.62 <i>Source Code</i> Kontrol Perulangan <i>For-Else</i>	44
Gambar 1.63 <i>Output</i> Kontrol Perulangan <i>For-Else</i>	44
Gambar 1.64 <i>Output</i> Kontrol Perulangan <i>For-Else</i>	44
Gambar 2.1 <i>Source Code</i> Mengakses data pada <i>List</i>	45
Gambar 2.2 Output mengakses data pada List.....	46
Gambar 2.3 <i>Source Code</i> <i>List-Akses</i>	46
Gambar 2.4 <i>Output</i> <i>List-Akses</i>	46
Gambar 2.5 <i>Source Code</i> <i>List-Ganti</i>	47
Gambar 2.6 <i>Output</i> <i>List-Ganti</i>	47
Gambar 2.7 <i>Source Code</i> <i>List-Tambah</i>	47
Gambar 2.8 <i>Output</i> <i>List-Tambah</i>	47
Gambar 2.9 <i>Source Code</i> <i>List-Hapus</i>	48
Gambar 2.10 <i>Output</i> <i>List-Hapus</i>	48
Gambar 2.11 <i>Source Code</i> <i>List-Combine</i>	48
Gambar 2.12 <i>Output</i> <i>List-Combine</i>	48
Gambar 2.13 <i>Source Code</i> <i>List</i> awal ke akir	49
Gambar 2.14 <i>Output</i> <i>List-Sort</i> dari awal ke akhir.....	49
Gambar 2.15 <i>Source Code</i> <i>List-Sort</i> dari akhir ke awal	49
Gambar 2.16 <i>Output</i> <i>List-Sort</i> dari akhir ke awal.....	49
Gambar 2.17 <i>Source Code</i> <i>Tuple-Satu</i> Tipe	50
Gambar 2.18 <i>Output</i> <i>Tuple-Satu</i> Tipe.....	50
Gambar 2.19 <i>Source Code</i> <i>Tuple-Banyak</i> Tipe	51
Gambar 2.20 <i>Output</i> <i>Tuple-Banyak</i> Tipe.....	51
Gambar 2.21 <i>Source Code</i> <i>Tuple</i> Akses-Indeks	51
Gambar 2.22 <i>Output</i> <i>Tuple</i> Akses-Indeks	52
Gambar 2.23 <i>Source Code</i> <i>Iuple-</i> Akses <i>Unpacking</i>	52
Gambar 2.24 <i>Output</i> <i>Iuple-</i> Akses <i>Unpacking</i>	52
Gambar 2.25 <i>Source Code</i> <i>Tuple-Concatenation</i>	53
Gambar 2.26 <i>Output</i> <i>Tuple-Concatenation</i>	53
Gambar 2.27 <i>Source Code</i> <i>Tuple-Slicing</i>	53
Gambar 2.28 <i>Output</i> <i>Tuple-Slicing</i>	54

Gambar 2.29 <i>Source Code Tuple-Delete</i>	54
Gambar 2.30 <i>Output Tuple-Delete</i>	54
Gambar 2.31 <i>Source Code Tuple-Sort</i>	55
Gambar 2.32 <i>Output Tuple-Sort</i>	55
Gambar 2.33 <i>Source Code Set-Add</i>	56
Gambar 2.34 <i>Output Set-Add</i>	56
Gambar 2.35 <i>Source Code Set-Update</i>	56
Gambar 2.36 <i>Output Set-Update</i>	57
Gambar 2.37 <i>Source Code Set-Delete</i>	57
Gambar 2.38 <i>Output Set-Delete</i>	57
Gambar 2.39 <i>Source Code Set-Combine</i>	58
Gambar 2.40 <i>Output Set-Combine</i>	58
Gambar 2.41 <i>Source Code Dictionary-Deklarasi 1</i>	59
Gambar 2.42 <i>Output Dictionary-Deklarasi 1</i>	59
Gambar 2.43 <i>Source Code Dictionary-Deklarasi 2</i>	60
Gambar 2.44 <i>Output Dictionary-Deklarasi 2</i>	60
Gambar 2.45 <i>Source Code Dictionary-tambah dan Update</i>	61
Gambar 2.46 <i>Dictionary-tambah dan Update</i>	61
Gambar 2.47 <i>Source Code Dictionary- Akses Elemen</i>	62
Gambar 2.48 <i>Output Dictionary- Akses Elemen</i>	62
Gambar 2.49 <i>Source Code Dictionary- Akses Get</i>	62
Gambar 2.50 <i>Output Dictionary- Akses Get</i>	63
Gambar 2.51 <i>Source Code Dictionary-Delete-del</i>	63
Gambar 2.52 <i>Output Dictionary-Delete-del</i>	63
Gambar 2.53 <i>Code Dictionary- Delete-pop</i>	64
Gambar 2.54 <i>Output Dictionary- Delete-pop</i>	64
Gambar 2.55 <i>Source Code Dictionary- Delete-popitem</i>	65
Gambar 2.56 <i>Output Dictionary- Delete-popitem</i>	65
Gambar 2.57 <i>Source Code Dictionary- Delete-clear</i>	65
Gambar 2.58 <i>Output Dictionary- Delete-clear</i>	65
Gambar 3.1 Deklarasi Sebuah Fungsi	66
Gambar 3.2 Pemanggilan Fungsi	66
Gambar 3.3 <i>Source Code Fungsi dengan Return</i>	67
Gambar 3.4 <i>Output Fungsi dengan Return</i>	67
Gambar 3.5 <i>Code Fungsi tanpa Return</i>	68
Gambar 3.6 <i>Output Fungsi tanpa Return</i>	68
Gambar 3.7 <i>Source Code Fungsi – Variabel Lokal</i>	69
Gambar 3.8 <i>Output Fungsi – Variabel Lokal</i>	69
Gambar 3.9 <i>Source Code Fungsi – Variabel Global</i>	70
Gambar 3.10 <i>Output Fungsi – Variabel Global</i>	70
Gambar 3.11 <i>Source Code Fungsi – Merubah Nilai Variabel Global</i>	71
Gambar 3.12 <i>Output Fungsi – Merubah Nilai Variabel Global</i>	71
Gambar 3.13 <i>Source Code Class – Contoh 1</i>	72
Gambar 3.14 <i>Output Class – Contoh 1</i>	72
Gambar 3.15 <i>Source Code Class – Contoh 2</i>	73
Gambar 3.16 <i>Output Class – Contoh 2</i>	73
Gambar 3.17 <i>Class – Parameter Self</i>	74
Gambar 3.18 <i>Class – Pembuatan Objek</i>	74

Gambar 3.19 Class – Pemanggilan Objek.....	74
Gambar 3.20 Inheritance - Syntax.....	75
Gambar 3.21 Contoh <i>Inheritance</i>	75
Gambar 3.22 Konsep <i>Inheritance</i>	76
Gambar 3.23 Contoh <i>calling Inheritance</i>	76
Gambar 3.24 <i>Sourcecode Inheritance</i>	77
Gambar 3.25 <i>Output Inheritance</i>	77
Gambar 3.26 Access Modifier <i>Public</i>	77
Gambar 3.27 Access Modifier <i>Protected</i>	78
Gambar 3.28 Access Modifier <i>Private</i>	78
Gambar 3.29 Konsep <i>single inheritance</i>	79
Gambar 3.30 <i>Source Code single Inheritance</i>	79
Gambar 3.31 <i>Output single Inheritance</i>	79
Gambar 3.32 Konsep <i>Multiple Inherintence</i>	80
Gambar 3.33 <i>Source Code Multiple Inheritance</i>	80
Gambar 3.34 <i>Output Multiple Inheritance</i>	80
Gambar 3.35 Konsep <i>Multilevel Inherintence</i>	81
Gambar 3.36 <i>Source Code Multilevel Inheritance</i>	82
Gambar 3.37 <i>Output Multilevel Inheritance</i>	82
Gambar 3.38 Konsep <i>Hierarchical inheritance</i>	82
Gambar 3.39 <i>Sourcecode Hierarchical Inheritance</i>	83
Gambar 3.40 <i>Output Hierarchical Inheritance</i>	83
Gambar 3.41 <i>Hierarchical inheritance</i>	83
Gambar 3.42 <i>Source Code Hybrid Inheritance</i>	84
Gambar 3.43 <i>Output Hybrid Inheritance</i>	84
Gambar 4.1 <i>Source Code Polymorphism – Overriding</i>	86
Gambar 4.2 <i>Output Program Polymorphism – Overriding</i>	86
Gambar 4.3 Source Code Polymorphism - Overloading Error	87
Gambar 4.4 Output Program Polymorphism - Overloading Error.....	87
Gambar 4.5 Code Polymorphism - Overloading Solution	88
Gambar 4.6 Output Program Polymorphism - Overloading Solution.....	88
Gambar 4.7 Source Code Polymorphism - Overloading Solution	89
Gambar 4.8 Output Program Polymorphism - Overloading Solution.....	89
Gambar 4.9 Enkapsulasi.....	90
Gambar 4.10 Source Code Implementasi Enkapsulasi	90
Gambar 4.11 Output Program Implementasi Enkapsulasi	90
Gambar 5.1 Perintah <i>install mysql-connector</i>	91
Gambar 5.2 Proses <i>install mysql-connector</i>	92
Gambar 5.3 <i>Source code</i> Membuat koneksi MySQL	92
Gambar 5.4 <i>Output</i> Membuat koneksi MySQL	93
Gambar 5.5 <i>code</i> Membuat Database	93
Gambar 5.6 <i>Output</i> Membuat Database.....	93
Gambar 5.7 <i>Output</i> Cek Database	94
Gambar 5.8 <i>Source code</i> Implementasi <i>Create</i>	95
Gambar 5.9 Implementasi <i>Create</i>	96
Gambar 5.10 <i>Source code</i> Implementasi <i>Insert</i>	97
Gambar 5.11 <i>Output Insert</i>	97
Gambar 5.12 <i>Output</i> Implementasi <i>Insert</i>	98

Gambar 5.13 <i>Source code</i> Implementasi <i>Read</i>	99
Gambar 5.14 <i>Output</i> Implementasi <i>Read</i>	99
Gambar 5.15 <i>Source code</i> Implementasi <i>Update</i>	100
Gambar 5.16 <i>Output Update</i>	100
Gambar 5.17 <i>Output</i> Implementasi <i>Update</i>	101
Gambar 5.18 <i>Source code</i> Implementasi <i>Delete</i>	102
Gambar 5.19 <i>Output Delete</i>	102
Gambar 5.20 <i>Output</i> Implementasi <i>Delete</i>	102
Gambar 6.1 <i>Source Code Requests.get</i>	104
Gambar 6.2 <i>Output Program Requests.get</i>	104
Gambar 6.3 <i>Code Inisiasi BeautifulSoup</i>	105
Gambar 6.4 <i>Program Inisiasi BeautifulSoup</i>	105
Gambar 6.5 Contoh HTML 1	106
Gambar 6.6 <i>Source Code Penggunaan Metode find()</i>	106
Gambar 6.7 <i>Program Penggunaan Metode find()</i>	106
Gambar 6.8 <i>Source Code Penggunaan Metode find_all()</i>	107
Gambar 6.9 <i>Output Program Penggunaan Metode find_all()</i>	107
Gambar 6.10 <i>Source Code Penggunaan Metode get_text()</i>	107
Gambar 6.11 <i>Output Program Penggunaan Metode get_text()</i>	107
Gambar 6.12 <i>Source Code Web Scraping Sederhana</i>	108
Gambar 6.13 <i>Output Program Web Scraping Sederhana</i>	108
Gambar 6.14 <i>Source Code Web Scraping Tabel</i>	109
Gambar 6.15 <i>Output Program Web Scraping</i>	109
Gambar 6.16 <i>Source Code pywinauto notepad</i>	110

DAFTAR TABEL

Tabel 1.1 Dasar <i>Print Statement</i>	20
Tabel 1.2 Tipe Data Dasar - <i>Numbers</i>	21
Tabel 1.3 Tipe Data Dasar - <i>Strings</i>	22
Tabel 1.4 Tipe Data Dasar – <i>Strings Assignment</i>	23
Tabel 1.5 Tipe Data Dasar – <i>String Slicing</i>	23
Tabel 1.6 Tipe Data Dasar – <i>String Concatenation</i>	24
Tabel 1.7 Operator Aritmetika – Penjumlahan dan Pengurangan	25
Tabel 1.8 Operator Aritmetika – Perkalian dan Pembagian.....	25
Tabel 1.9 Operator Aritmetika – <i>Floored Division</i>	26
Tabel 1.10 Operator Aritmetika – <i>Modulo</i>	27
Tabel 1.11 Operator Perbandingan.....	27
Tabel 1.12 <i>Input</i>	28
Tabel 1.13 <i>Input</i> – Multiplikasi.....	29
Tabel 1.14 <i>Import Module</i>	30
Tabel 1.15 <i>Import From Module</i>	31
Tabel 1.16 <i>Import from Module As</i>	32
Tabel 1.17 Seleksi Kondisional <i>If</i>	33
Tabel 1.18 Struktur Kondisional <i>If-Else</i>	34
Tabel 1.19 Struktur Kondisional <i>if-elif-else</i>	36
Tabel 1.20 Perulangan <i>For</i>	37
Tabel 1.21 Perulangan <i>While</i>	38
Tabel 1.22 Perulangan Bertingkat.....	39
Tabel 1.23 Kontrol Perulangan <i>Break</i>	40
Tabel 1.24 Kontrol Perulangan <i>Continue</i>	41
Tabel 1.25 Kontrol Perulangan <i>Pass</i>	42
Tabel 1.26 Kontrol perulangan <i>While-Else</i>	43
Tabel 1.27 Kontrol perulangan <i>For-Else</i>	44
Tabel 2.1 Mengakses <i>List</i>	45
Tabel 2.2 <i>List</i> -Akses.....	46
Tabel 2.3 <i>List</i> -Ganti	47
Tabel 2.4 <i>List</i> -Tambah	47
Tabel 2.5 <i>List</i> -Hapus	48
Tabel 2.6 <i>List</i> -Combine	48
Tabel 2.7 <i>List</i> -Sort dari awal ke akhir	49
Tabel 2.8 <i>List</i> -Sort dari akhir ke awal	49
Tabel 2.9 <i>Tuple</i> -Satu Tipe	50
Tabel 2.10 <i>Tuple</i> -Banyak Tipe	50
Tabel 2.11 <i>Tuple</i> Akses Indeks	51
Tabel 2.12 <i>Tuple</i> -Akses <i>Unpacking</i>	52
Tabel 2.13 <i>Tuple</i> - <i>Concatenation</i>	52
Tabel 2.14 <i>Tuple</i> - <i>Slicing</i>	53
Tabel 2.15 <i>Tuple</i> - <i>Delete</i>	54
Tabel 2.16 <i>Tuple</i> - <i>Sort</i>	55
Tabel 2.17 <i>Set</i> - <i>Add</i>	56
Tabel 2.18 <i>Set</i> - <i>Update</i>	56
Tabel 2.19 <i>Set</i> - <i>Delete</i>	57
Tabel 2.20 <i>Set</i> - <i>Combine</i>	58

Tabel 2.21 <i>Dictionary</i> -Deklarasi 1	58
Tabel 2.22 <i>Dictionary</i> -Deklarasi 2	59
Tabel 2.23 <i>Dictionary</i> - Menambah dan Update Elemen.....	60
Tabel 2.24 <i>Dictionary</i> - Akses Key	61
Tabel 2.25 <i>Dictionary</i> - Akses Get.....	62
Tabel 2.26 <i>Dictionary-Delete-del</i>	63
Tabel 2.27 <i>Dictionary- Delete-pop</i>	64
Tabel 2.28 <i>Dictionary- Delete-popitem</i>	64
Tabel 2.29 <i>Dictionary- Delete-clear</i>	65
Tabel 3.1 Fungsi - dengan <i>Return</i>	67
Tabel 3.2 Fungsi - tanpa <i>Return</i>	67
Tabel 3.3 Fungsi – Variabel Lokal	68
Tabel 3.4 Fungsi – Variabel Global	69
Tabel 3.5 Fungsi – Merubah Nilai Variabel Global	70
Tabel 3.6 Class – Contoh 1	72
Tabel 3.7 Class – Contoh 2	73
Tabel 3.8 <i>Inheritance</i>	76
Tabel 3.9 Contoh <i>Single Inheritance</i>	79
Tabel 3.10 Contoh <i>Multiple Inheritance</i>	80
Tabel 3.11 Contoh <i>Multilevel Inheritance</i>	81
Tabel 3.12 Contoh <i>Hierarchical Inheritance</i>	83
Tabel 3.13 Contoh <i>Hybrid Inheritance</i>	83
Tabel 4.1 <i>Polymorphism – Overriding</i>	85
Tabel 4.2 <i>Polymorphism - Overloading Error</i>	87
Tabel 4.3 <i>Polymorphism - Overloading Solution</i>	87
Tabel 4.4 Kelas Abstrak	89
Tabel 4.5 Implementasi Enkapsulasi.....	90
Tabel 5.1 Koneksi MySQL	92
Tabel 5.2 Membuat Database	93
Tabel 5.3 Cek Database.....	94
Tabel 5.4 <i>Create - Database</i>	95
Tabel 5.5 <i>Insert - Database</i>	96
Tabel 5.6 <i>Output</i> Implementasi <i>Insert</i>	97
Tabel 5.7 <i>Read - Database</i>	98
Tabel 5.8 <i>Update - Database</i>	100
Tabel 5.9 <i>Delete - Database</i>	101
Tabel 6.1 Fungsi <i>Request.get</i>	104
Tabel 6.2 Inisiasi <i>Beautifulsoup</i>	105
Tabel 6.3 Metode <i>BeautifulSoup</i>	105
Tabel 6.4 Fungsi <i>find()</i> <i>BeautifulSoup</i>	106
Tabel 6.5 Fungsi <i>find_all()</i> <i>BeautifulSoup</i>	107
Tabel 6.6 Fungsi <i>get_text()</i> <i>BeautifulSoup</i>	107
Tabel 6.7 <i>Web Scraping</i> Sederhana	108
Tabel 6.8 <i>Web Scraping</i> pada Tabel	109
Tabel 6.9 Pywinauto <i>notepad</i>	110

Modul 1: Pengenalan dan Dasar Python

1.1 Tujuan Praktikum:

1. Memahami tipe data dan operator dalam python
2. Mengetahui cara menginput data dari keyboard user
3. Mengetahui konsep logika dan seleksi kondisional
4. Mengetahui sintaks dan logika perulangan
5. Menerapkan seleksi kondisional dan perulangan

1.2 Alat dan Bahan:

1. PC yang terinstall Python 3
2. PC yang terinstall IDE Visual Studio Code / PyCharm

1.3 Dasar Teori

1.3.1 Sejarah Python

Python adalah bahasa pemrograman multifungsi yang dibuat oleh Guido van Rossum dan dirilis pada tahun 1991. Python diciptakan untuk menjadi interpreter yang memiliki kemampuan penanganan kesalahan (*exception handling*) dan mengutamakan sintaksis yang mudah dibaca serta dimengerti (*readability*). Didesain untuk memudahkan dalam prototyping, Python menjadi bahasa yang sangat mudah dipahami dan fleksibel.

Karena didesain untuk memudahkan dalam prototyping, maka python memiliki karakteristik yang berbeda seperti penggunaan indentasi, dan penggunaan titik-koma yang bersifat opsional. Pada kode pemrograman python, titik-koma sangat jarang ditemui atau hampir tidak ada karena dianggap non-pythonic way oleh komunitasnya.

1.3.2 Aplikasi dan Penggunaan Python

Python menganut konsep pemrograman multi paradigma, maksudnya adalah pengembang dapat menggunakan python dengan paradigma prosedural/fungsional maupun paradigma berorientasi objek.

Python juga dapat digunakan untuk mengembangkan software dan script di berbagai bidang sehingga membuat popularitas dan penggunaanya meningkat pesat, beberapa bidang yang dapat dijangkau Python:

- a. Server-Side Web Development
- b. Internet of Things
- c. Network Automation
- d. Data Science
- e. System Scripting
- f. Penetration Testing Tools

Selain kegunaan yang sangat banyak, Python juga memiliki package manager pip dan sumber package yang lengkap, yang ditunjang oleh komunitas, yang berarti siapa saja dapat menuliskan package dan berkontribusi terhadap python untuk memudahkan pengembang lainnya.

1.3.3 Konsep Dasar Pemrograman Python

A. IDE

Untuk mulai menuliskan kode python dan menjalankannya sebenarnya kita hanya membutuhkan text editor dan python interpreter. Namun untuk menunjang pelaksanaan praktikum kali ini dan meningkatkan produktivitas sebagai Software Engineer, maka kita akan menggunakan IDE Visual Studio Code.

Integrated Development Environment (IDE) lebih dari sekedar text editor untuk membuat kode, di dalamnya tergabung juga berbagai fasilitas development misalnya Code Versioning, Interpreter, Visualization dll. Instruksi untuk meng-install IDE Visual Studio Code dan PyCharm terdapat di modul instalasi IDE, yang dapat diunduh disini.

B. Dasar Python

a. Print Statement

Pada bahasa pemrograman Python, untuk menampilkan variable, text ataupun hasil perhitungan kita dapat menggunakan fungsi print() atau biasanya disebut print statement.

Catatan: Jangan khawatir jika ada kode yang belum dimengerti karena akan dibahas lagi pada lanjutan modul ini.

Tabel 1.1 Dasar Print Statement

SOURCE CODE	OUTPUT
 <p>Gambar 1.1 Source Code Print Statement</p>	 <p>Gambar 1.2 Source Code Print Statement</p>

Dari kode di atas, ada satu hal lagi yang bisa kita simpulkan, yaitu print statement akan langsung menuju baris baru untuk setiap

perintahnya, berbeda dengan printf pada C yang harus di berikan tanda “\n” atau identifier newlinenya.

b. Tipe Data Dasar

Python memiliki pendekatan yang berbeda untuk mendefinisikan data, jika dalam bahasa pemrograman lainnya kita akan mendefinisikan tipe datanya saat deklarasi variable, di Python kita tidak akan melakukan hal tersebut, hal ini dikarenakan Python menganut sistem Dynamic Typing dan akan langsung mengenali tipe data yang kita isikan ke dalam variable.

Lalu bagaimana kita bisa mengetahui tipe data yang dikenali oleh Python? Untuk permasalahan tersebut kita dapat menggunakan fungsi type().

1. Tipe Data Dasar – Numbers

Pada python terdapat dua tipe data dasar yang berhubungan dengan angka dan sangat sering ditemui yaitu float dan int. Float atau floating point adalah bilangan desimal, sedangkan int atau integer adalah bilangan bulat

Tabel 1.2 Tipe Data Dasar - *Numbers*

SOURCE CODE
<pre>● ● ● bilangan_desimal = 1.25 bilangan_bulat = 10 print("Tipe data", bilangan_desimal, "adalah", type(bilangan_desimal)) print("Tipe data", bilangan_bulat, "adalah", type(bilangan_bulat))</pre>
Gambar 1.3 Source Code Tipe Data Dasar - <i>Numbers</i>

OUTPUT
<pre>Tipe data 1.25 adalah <class 'float'> Tipe data 10 adalah <class 'int'></pre>
Gambar 1.4 Source Code Tipe Data Dasar - <i>Numbers</i>

Tanpa deklarasi tipe data, python dapat langsung mengenali tipe data yang kita gunakan dalam variable bilangan_bulat yang dikenali sebagai int dan bilangan_desimal yang dikenali sebagai float.

2. Tipe Data Dasar – Strings

Selain tipe data yang berkaitan dengan angka, Python juga dapat mengenali tipe data yang terdiri dari rangkaian karakter unikode, pada python tipe data ini masuk kedalam kelas str atau Strings.

Tabel 1.3 Tipe Data Dasar - *Strings*

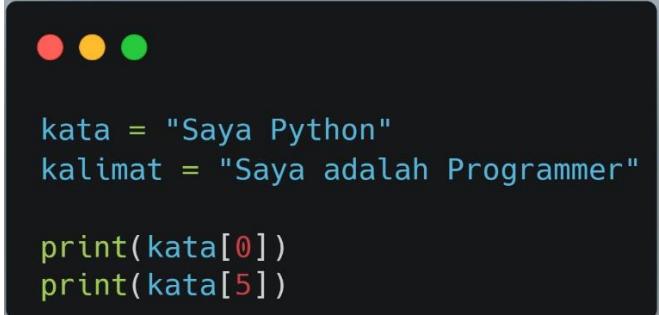
SOURCE CODE		
<pre>kata = "Saya" kalimat = "Saya adalah programmer" kalimat_multibaris = """ Ini adalah blog singkat saya untuk menunjukkan multiline strings pada python """ print(type(kata)) print(type(kalimat)) print(type(kalimat_multibaris))</pre>		
<p>Gambar 1.5 <i>Source Code</i> Tipe Data Dasar - <i>String</i></p> <th>OUTPUT</th> <td><pre><class 'str'> <class 'str'> <class 'str'></pre></td>	OUTPUT	<pre><class 'str'> <class 'str'> <class 'str'></pre>

Karakteristik dari Strings adalah, tipe data ini selalu dibungkus dalam tanda “ “ atau ‘ ‘ . Jika kita menaruh angka diantara tanda ini, maka tipe datanya akan berubah menjadi strings, bukan integer maupun float.

3. Strings – Operasi pada strings

Karena strings merupakan rangkaian karakter Unicode, maka strings dapat diakses seperti array menggunakan indeks, namun kita tidak dapat melakukan assignment terhadap index tertentu, perhatikan kode berikut:

Tabel 1.4 Tipe Data Dasar – *Strings Assignment*

SOURCE CODE
 <pre>kata = "Saya Python" kalimat = "Saya adalah Programmer" print(kata[0]) print(kata[5])</pre>
Gambar 1.7 <i>Source Code</i> Tipe Data Dasar – <i>String Assignment</i>

OUTPUT
 S P

Gambar 1.8 *Output* Tipe Data Dasar – *String Assignment*

Selain mengakses nilai pada suatu index, strings di python juga dapat di slice dengan cara mengisikan nilai mulai dan nilai berhenti, namun karakter pada nilai berhenti tidak akan dimasukkan dalam hasil slicing nya, atau bersifat excluded.

Contoh String Slicing:

Tabel 1.5 Tipe Data Dasar – *String Slicing*

SOURCE CODE	OUTPUT
 <pre>kata = "Saya Python" print(kata[0:4])</pre>	 Saya Gambar 1.10 <i>Output Print Statement</i>

Operasi lainnya yang bisa dilakukan terhadap strings adalah String Concatenation, operasi ini ditujukan untuk menggabungkan beberapa strings menjadi satu, seperti contoh berikut:

Tabel 1.6 Tipe Data Dasar – *String Concatenation*

SOURCE CODE
<pre>nama_awal = "Software Engineering" nama_akhir = " and Applications" nama_lengkap = nama_awal + nama_akhir</pre>
<p>Gambar 1.11 <i>Source Code</i> Tipe Data Dasar – <i>String Concatenation</i></p>
OUTPUT
<pre>Software Engineering and Applications</pre>
<p>Gambar 1.12 <i>Output</i> Tipe Data Dasar – <i>String Concatenation</i></p>

Khusus untuk strings, operasi + akan menyebabkan string yang berada di sebelah kiri operator + digabungkan dengan string yang berada di sebelah kanan operator.

C. Operator – Operator Aritmatika

a. Operasi Penjumlahan dan Pengurangan

Tabel 1.7 Operator Aritmetika – Penjumlahan dan Pengurangan

SOURCE CODE
<pre>● ● ● x = 4 y = 2 #operasi penjumlahan z = x + y print("Hasil dari ",x,"+",y,"=",z) #operasi pengurangan z = x - y print("Hasil dari ",x,"-",y,"=",z)</pre>

Gambar 1.13 *Source Code* Operator Aritmetika – Penjumlahan dan Pengurangan

OUTPUT
<pre>Hasil dari 4 + 2 = 6 Hasil dari 4 - 2 = 2</pre>

Gambar 1.14 *Output* Operator Aritmetika – Penjumlahan dan Pengurangan

b. Operasi Perkalian dan Pembagian

Tabel 1.8 Operator Aritmetika – Perkalian dan Pembagian

SOURCE CODE
<pre>● ● ● x = 4 y = 2 #operasi penjumlahan z = x * y print("Hasil dari ",x,"*",y,"=",z) #operasi pengurangan z = x / y print("Hasil dari ",x,"/",y,"=",z)</pre>

Gambar 1.15 *Source Code* Operator Aritmetika – Perkalian dan Pembagian

```
Hasil dari 4 * 2 = 8
Hasil dari 4 / 2 = 2.0
```

Gambar 1.16 *Output* Operator Aritmetika – Perkalian dan Pembagian

Dari contoh kode dan hasil running diatas, terdapat satu kesimpulan yaitu operasi pembagian / division pada python akan mengembalikan nilai berupa float.

Mengapa begitu?

Karena Python secara default ingin membuat penggunaanya terhindar dari kesalahan perhitungan, bayangkan jika hasilnya bilangan bulat, maka $5/2$ akan menghasilkan 2, yang seharusnya nilainya 2.5, jika perhitungannya membutuhkan hasil yang presisi, ini tentu akan menghasilkan error secara matematis yang sangat besar. Jika anda membutuhkan hasil perhitungannya berupa integer atau dibulatkan ke bilangan bulat terdekat, maka operator yang digunakan adalah `//`, penggunaannya sebagai berikut:

c. Operasi Floored Division

Tabel 1.9 Operator Aritmetika – *Floored Division*

SOURCE CODE

```
● ● ●

x = 4
y = 2

#operasi penjumlahan
z = x + y
print("Hasil dari ",x,"*",y,"=",z)

#operasi pengurangan
z = x - y
print("Hasil dari ",x,"/",y,"=",z)
```

Gambar 1.17 *Source Code* Operator Aritmetika – *Floored Division*

OUTPUT

```
Hasil dari 7 // 2 = 3
```

Gambar 1.18 *Output* Operator Aritmetika – *Floored Division*

Pada operasi Floored Division, hasil bagi yang berupa bilangan desimal akan diubah menjadi bilangan bulat terdekat, pada contoh diatas, jika menggunakan operasi pembagian biasa, maka hasilnya

adalah 3.5, namun karena menggunakan floored division maka hasil dari $7//2$ adalah 3.

d. Operasi Sisa Hasil Bagi (Modulo)

Tabel 1.10 Operator Aritmetika – *Modulo*

SOURCE CODE
<pre>● ● ● a = 5 b = 2 c = a % b #operasi modulus print("Hasil dari ",a,"%",b,"=",c)</pre>
OUTPUT
<pre>Hasil dari 5 % 2 = 1</pre>

Gambar 1.19 Source Code Operator Aritmetika – *Modul*

Gambar 1.20 Output Operator Aritmetika – *Modulo*

e. Operator – Operator Perbandingan

Operator perbandingan adalah operator yang akan membandingkan dua nilai, biasanya operasi ini akan digunakan saat akan membuat program yang memerlukan pengambilan keputusan berdasarkan kondisi tertentu (subjek ini akan dibahas lebih lanjut di modul berikutnya), operasi perbandingan ini akan menghasilkan nilai “True” jika kondisi terpenuhi atau “False” jika kondisi tidak terpenuhi.

Contoh Penggunaan Operator Perbandingan:

Tabel 1.11 Operator Perbandingan

SOURCE CODE	OUTPUT
<pre>● ● ● x = 5 y = 2 print(x < y) #Output False print(x > y) #Output True print(x <= y) #Output False print(x >= y) #Output True print(x != y) #Output True print(x == y) #Output False</pre>	<pre>False True False True True False</pre>

Gambar 1.21 Source Code Operator Perbandingan

Gambar 1.22 Output Operator Perbandingan

D. Input

Pada program komputer, ada saatnya program akan membutuhkan input dari user, misalnya program untuk menghitung luas segitiga tertentu, maka akan membutuhkan input alas dan tinggi yang nantinya akan dihitung oleh komputer untuk perhitungan luasnya. Untuk mencapai hal ini dengan Python, sudah disediakan fungsi built-in **input()**.

Fungsi ini akan menerima parameter, yang nantinya akan dijadikan prompt saat menunggu inputan dari user.

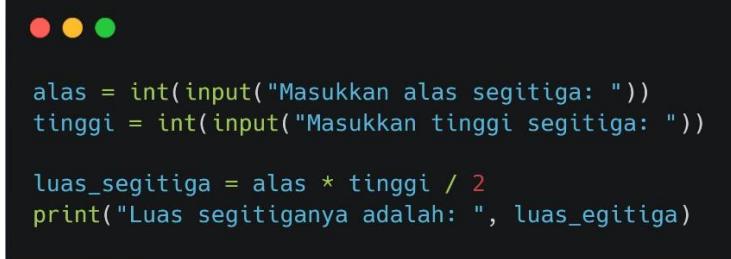
Namun, pada python, tipe data yang akan diterima secara default oleh fungsi **input()** adalah str, sehingga kita harus mengubah terlebih dahulu data inputan secara eksplisit, dalam bahasa pemrograman lain, metode ini biasanya disebut dengan typecasting, untuk mengubah tipe data dari suatu variable, kita bisa melakukannya dengan beberapa fungsi, tergantung tipe data tujuan nya, berikut beberapa fungsinya:

Mengubah ke integer : variable = int(variable)
Mengubah ke float : variable = float(variable)
Mengubah ke strings : variable = str(variable)

Untuk input kita dapat langsung mengubah nya dengan cara menjadikan fungsi input sebagai parameter dari fungsi diatas, contohnya:

umur = int(input("Masukkan umur anda : "))
Contoh program dengan input (Menghitung luas segitiga):

Tabel 1.12 *Input*

SOURCE CODE
 <pre>● ● ● alas = int(input("Masukkan alas segitiga: ")) tinggi = int(input("Masukkan tinggi segitiga: ")) luas_segitiga = alas * tinggi / 2 print("Luas segitiganya adalah: ", luas_segitiga)</pre>

Gambar 1.23 *Source Code Input*

OUTPUT	
<p>Pada saat dijalankan, maka program akan menunggu inputan terlebih dahulu dari user</p> <p>Masukkan alas segitiga: </p> <p>Gambar 1.24 <i>Input – menunggu input user</i></p>	<p>Jika kedua nilai sudah diinput maka akan muncul hasil perhitungannya</p> <p>Masukkan alas segitiga: 3 Masukkan tinggi segitiga: 5 Luas segitiganya adalah: 7.5</p> <p>Gambar 1.25 <i>Input – Hasil</i></p>

Namun jika tipe data inputan tidak diubah terlebih dahulu, saat melakukan perhitungan maka akan muncul error seperti berikut:

```
Masukkan alas segitiga: 10
Masukkan tinggi segitiga: 20
Traceback (most recent call last):
  File "d:\Kuliah\Season 5\SEA\Modul 1 SEA\ngoding.py", line 4, in <module>
    luas_segitiga = alas * tinggi / 2
TypeError: can't multiply sequence by non-int of type 'str'
```

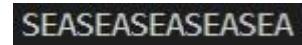
Gambar 1.26 *Input – Failed*

Python akan memberikan pesan error dengan kategori `TypeError`, yang berarti tipe data kita tidak dapat dioperasikan. Pesannya secara spesifik adalah “**“can’t multiply sequence by non-int of type ‘str”**”, maksud dari pesan diatas adalah, python tidak dapat melakukan operasi perkalian dengan inputan tipe data keduanya str.

Namun, operasi multiplikasi atau perkalian pada python dapat digunakan jika tipe datanya adalah str dengan int, ini akan menghasilkan perulangan terhadap string, suatu fitur yang tidak dapat dilakukan banyak bahasa pemrograman pada umumnya, contohnya:

Tabel 1.13 *Input – Multiplikasi*

SOURCE CODE

 Gambar 1.27 <i>Source Code</i> Input – Multiplikasi
OUTPUT  Gambar 1.28 <i>Input</i> – Multiplikasi

E. Module

Dalam proses membuat sebuah program ataupun software, kita seringkali membutuhkan fungsi tertentu, sebagai contoh, kita pasti membutuhkan beberapa fungsi seperti akar kuadrat dari sebuah bilangan ataupun kuadrat dari sebuah bilangan. Tentu saja kita dapat membuatnya sendiri, namun sebagai seorang Software Engineer, kita harus meminimalisir praktik re-invent the wheel, atau membuat kembali fungsi yang sudah ada.

Untuk mengatasi hal ini Python menyediakan fitur import yang berguna untuk memanggil library yang tersedia pada Python.

Mengapa library harus di-import dan tidak disediakan saja semuanya dari awal?

Hal ini bertujuan untuk mencegah ukuran program ataupun software menjadi terlalu besar, dengan fungsi import kita dapat memanggil module atau library yang kita butuhkan saja.

a. Import Module

Ada dua cara untuk memanggil fungsi dari module, yaitu dengan cara mengimport module pemilik fungsi tersebut, atau dengan cara mengambil suatu fungsi dari satu module tertentu

Tabel 1.14 *Import Module*

SOURCE CODE

```

import math #mengimport module math
angka = 25 #menginisialisasikan nilai angka dengan 25
kuadrat_dari_angka = math.sqrt(25) # memanggil fungsi #sqrt dari module #math dan memberi nilai 25 ke #parameternya
#fungsi sqrt adalah singkatan dari square root yang artinya adalah #akar pangkat 2
print(kuadrat_dari_angka) # baris ini akan menghasilkan 5.0

```

Gambar 1.29 *Source Code Import Module*

OUTPUT

5.0

Gambar 1.30 *Output Import Module*

b. Import From Module

Untuk mengatasi pemborosan memory ketika kita hanya menggunakan beberapa fungsi dari suatu module, kita dapat menggunakan cara lain untuk memanggil fungsi yang dibutuhkan saja dari suatu module. Contoh:

Tabel 1.15 *Import From Module*

SOURCE CODE

```

from math import sqrt

angka = 25

kuadrat_dari_angka = sqrt(25)
print(kuadrat_dari_angka)

```

Gambar 1.31 *Source Code Import from Module*

OUTPUT

5.0

Gambar 1.32 *Output Import from Module*

Source code di atas sekilas terlihat sama saja dengan contoh sebelumnya, namun terdapat perbedaan yang jelas saat kita memanggil fungsinya. Di baris ke-5 untuk memanggil fungsi **sqrt** kita tidak lagi menuliskan **math.sqrt()** melainkan hanya **sqrt** saja, hal ini dikarenakan pada baris pertama, kita menggunakan sintaks berbeda untuk memanggil fungsi **sqrt**, yaitu:

```
from math import sqrt
```

Sintaks ini bertujuan untuk memanggil satu buah fungsi saja dari module tertentu, cara penggunaannya adalah:

```
from nama_module import fungsi
```

Dengan cara ini kita dapat menghemat banyak resource memory yang dapat meningkatkan performa dari program kita, cara ini lebih efisien jika kita hanya menggunakan sedikit saja fungsi dari module tertentu.

c. Import as

Pada contoh sebelumnya kita sudah berhasil menghemat memory dengan hanya menggunakan **sqrt** dari modul **math**, namun ada satu lagi fitur pada python dimana kita dapat memberi nama lain untuk module yang kita panggil.

Sebagai contoh, kita ingin memberi nama **kuadrat** kepada fungsi **sqrt**, kita dapat melakukannya dengan sintaks:

```
from math import sqrt as kuadrat
```

Sintaks diatas akan meng-import fungsi **sqrt** dari module **math**, dan memberi nama alias **kuadrat**, yang dapat kita gunakan untuk memanggil fungsi tersebut. Contoh:

Tabel 1.16 *Import from Module As*

SOURCE CODE

```
from math import sqrt as kuadrat

angka = 25

kuadrat_dari_angka = kuadrat(25)
print(kuadrat_dari_angka)
```

Gambar 1.33 *Source Code Import As*

OUTPUT

```
5.0
```

Gambar 1.34 *Output Import As*

F. Seleksi Kondisional dan Perulangan

Dalam Bahasa pemrograman ada yang namanya seleksi kondisional dan perulangan, di dalam sebuah proses memprogram, kita membutuhkan sebuah penyeleksian kondisi dan perulangan kode program, untuk tujuan tersebut sama seperti Bahasa pemrograman yang lainnya python juga memiliki sintak pemrograman yang tak jauh beda dengan Bahasa pemrograman lainnya.

Disini, kita akan membahas if, if-else, if-elif-else untuk seleksi kondisional dan untuk perulangan adalah for dan while. Selain itu, kita akan membahas kontrol perulangan,

a. Struktur Kondisional

1. If

Digunakan untuk melakukan pengujian suatu kondisi yang kita inginkan, cara penulisan sintaksnya:

```
If kondisi:  
statement
```

Kondisi adalah ekspresi nilai yang akan dibandingkan, jika kondisinya bernilai True maka statement akan dijalankan, dan jika kondisinya bernilai False maka statement tersebut tidak akan jalankan.

Tabel 1.17 Seleksi Kondisional *If*

SOURCE CODE

```

ip = float(input("Masukkan Nilai IP Anda :"))
if ip >= 3.5:
    print("Selamat Anda Lulus Cumlaude dengan IP Sebesar", ip)
print("Semangat terus ya...")

```

Gambar 1.35 *Source Code* Struktur Kondisional

OUTPUT	
Masukkan Nilai IP Anda :3.7 Selamat Anda Lulus Cumlaude dengan IP Sebesar 3.7 Semangat terus ya..	Masukkan Nilai IP Anda :3.4 Semangat terus ya..
Gambar 1.36 <i>Output</i> Struktur Kondisional <i>if-true</i>	Gambar 1.37 <i>Output</i> Struktur Kondisional <i>if-false</i>

Program di atas akan menentukan kita akan dinyatakan lulus Cumlaude atau tidak dari nilai IP yang kita miliki dengan kondisi jika nilai IP lebih dari sama dengan 3.5.

2. *If-else*

Kita dapat menentukan apa yang akan dilakukan apabila kondisi tersebut tidak terpenuhi dengan menggunakan keyword else. Bentuk penulisannya adalah:

```

If kondisi :
statement "true"

else:
statement "lainnya"

```

Contoh *Source Code*:

Tabel 1.18 Struktur Kondisional *If-Else*

SOURCE CODE

```

ip = float(input("Masukkan Nilai IP Anda :"))
if ip >= 3.5:
    print("Selamat Anda Lulus Cumlaude dengan IP Sebesar", ip)
else:
    print("Anda Lulus dengan IP sebesar", ip)
print("Semangat terus ya...")

```

Gambar 1.38 Source Code Struktur Kondisional *if-else*

OUTPUT	
<pre> Masukkan Nilai IP Anda :3.5 Selamat Anda Lulus Cumlaude dengan IP Sebesar 3.5 Semangat terus ya.. </pre> <p>Gambar 1.39 <i>Output</i> Struktur Kondisional <i>if-else</i> (<i>true</i>)</p>	<pre> Masukkan Nilai IP Anda :3.2 Anda Lulus dengan IP sebesar 3.2 Semangat terus ya.. </pre> <p>Gambar 1.40 <i>Output</i> Struktur Kondisional <i>if-else</i> (<i>false</i>)</p>

Dari kedua gambar diatas, hal yang membedakannya adalah Lulus dengan *Cumlaude* atau Lulus biasa, dimana ketika variabel ip memiliki nilai kurang dari 3.5 maka akan muncul *output* "Anda Lulus dengan IP sebesar 3.2", tetapi apabila variabel ip memiliki nilai lebih dari sama dengan 3.5 maka akan muncul *output* "Selamat Anda Lulus *Cumlaude* dengan IP Sebesar 3.5".

3. *If-elif-else*

Apabila kita ingin membuat program lebih dari satu kondisi kita bisa menggunakan elif, sintaks penulisannya:

```

if kondisi1:
    statement1 "true1"
elif kondisi2:
    statement2 "true2"
else:
    statement

```

Seleksi kondisional *if-elif-else* digunakan untuk memeriksa beberapa buah kondisi. Banyak digunakan untuk sebuah program dengan banyak kondisi/persyaratan. *Statement2* akan dijalankan jika kondisi1 tidak terpenuhi dan kondisi2 terpenuhi. Jika semua

kondisi tidak terpenuhi, maka program akan menjalankan *statement else*.

Contoh Source Code:

Tabel 1.19 Struktur Kondisional *if-elif-else*

SOURCE CODE
<pre>● ● ● ip = float(input("Masukkan Nilai IP Anda :")) if ip < 1.5: print("Anda Tidak Lulus") elif ip < 3.5: print("Selamat Anda Lulus dengan IP sebesar", ip) else: print("Selamat Anda Lulus Cumlaude dengan IP sebesar", ip) print("Tetap Semangat ya..")</pre>

Gambar 1.41 *Source Code* Struktur Kondisional *if-elif-else*

OUTPUT
<pre>Masukkan Nilai IP Anda :1.35 Anda Tidak Lulus Tetap Semangat ya..</pre>
<pre>Masukkan Nilai IP Anda :3.21 Selamat Anda Lulus dengan IP sebesar 3.21 Tetap Semangat ya..</pre>
<pre>Masukkan Nilai IP Anda :3.52 Selamat Anda Lulus Cumlaude dengan IP sebesar 3.52 Tetap Semangat ya..</pre>

Gambar 1.42 *Output if-elif-else (if true)*

Gambar 1.43 *Output if-elif-else (elif true)*

Gambar 1.44 *if-elif-else (else true)*

b. Perulangan

Perulangan data atau yang biasa disebut dengan *looping* adalah proses yang dilakukan secara berulang-ulang sampai batas yang ditentukan. Struktur perulangan digunakan untuk mengulang sekumpulan perintah sesuai dengan kondisi yang diberikan. Beberapa contoh perulangan pada *Python* adalah:

1. *For*

Struktur perulangan *for* biasa digunakan untuk mengulang suatu proses yang telah diketahui jumlah perulangannya, *statement* perulangan ini yang paling sering digunakan. Format penulisannya adalah sebagai berikut:

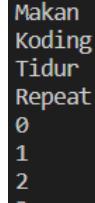
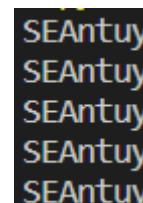
```
for ...(1) in ...(2):
```

Penjelasan:

1 sebagai variabel iterator (pengulang) dan 2 sebagai batas pengulangan.

Contoh *Source Code*:

Tabel 1.20 Perulangan *For*

SOURCE CODE	
 <pre>SEA=["Makan","Koding","Tidur","Repeat"] for item in SEA: print(item) for x in range (4): print(x)</pre>	 <pre>for y in range (5): print ("SEAntuy")</pre>
OUTPUT	
 <pre>Makan Koding Tidur Repeat 0 1 2 3</pre>	 <pre>SEAntuy SEAntuy SEAntuy SEAntuy SEAntuy</pre>
Gambar 1.45 <i>Output</i> Perulangan <i>for</i>	Gambar 1.48 <i>Output</i> Perulangan <i>for</i>

2. *While*

While adalah perulangan yang dimana dia akan membandingkan kondisinya, apabila kondisinya *True* (benar) maka perulangan akan terus dijalankan, perulangan akan berhenti apabila kondisi *False* (salah).

While (kondisi) :
Statement
Increment

Tabel 1.21 Perulangan While

SOURCE CODE
<pre>x=int(input("Banyaknya Perulangan : ")) y=0 while y<x: print ("Selamat Datang di Praktikum PBO") y=y+1</pre>
<pre>Banyaknya Perulangan : 5 Selamat Datang di Praktikum PBO Selamat Datang di Praktikum PBO</pre>

Gambar 1.49 Source Code Perulangan While

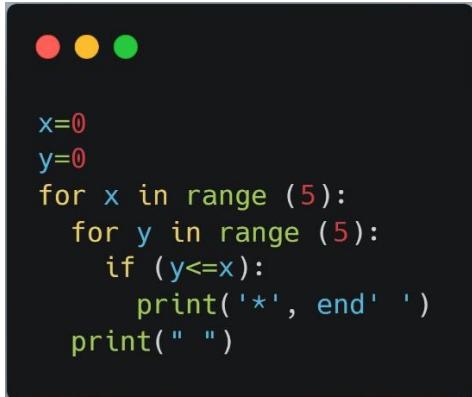
Gambar 1.50 Output Perulangan While

c. Perulangan Bertingkat

Perulangan bertumpuk secara sederhana dapat diartikan, terdapat satu atau lebih perulangan didalam sebuah perulangan. Banyaknya tingkatan perulangan tergantung dari kebutuhan. Perulangan luar, biasanya digunakan untuk mendefinisikan baris sedangkan perulangan dalam digunakan untuk mendefinisikan kolom. Contoh sintaksnya adalah:

For ... in ... :
for ... in ... :

Tabel 1.22 Perulangan Bertingkat

SOURCE CODE
 <pre>x=0 y=0 for x in range (5): for y in range (5): if (y<=x): print('*', end=' ') print(" ")</pre>
OUTPUT
 <pre>* * * * * * * * * * * * * * *</pre>

Gambar 1.51 *Source Code* Perulangan Bertingkat

Gambar 1.52 *Output* Perulangan Bertingkat

d. Kontrol Perulangan

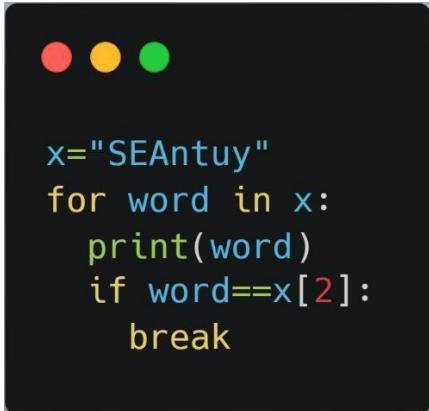
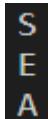
Kontrol perulangan adalah fungsi yang dapat membantu kita untuk mengatur jalannya perulangan.

1. *Break*

Break digunakan untuk menghentikan perulangan. Sintaks penulisannya:

```
For ... in ... :
    Statement 1
    If kondisi :
        break
```

Tabel 1.23 Kontrol Perulangan *Break*

SOURCE CODE
 <pre>x="SEAntuy" for word in x: print(word) if word==x[2]: break</pre>
OUTPUT
 <pre>S E A</pre>

Gambar 1.53 Source Code Kontrol Perulangan *Break*

Gambar 1.54 Output Kontrol Perulangan *Break*

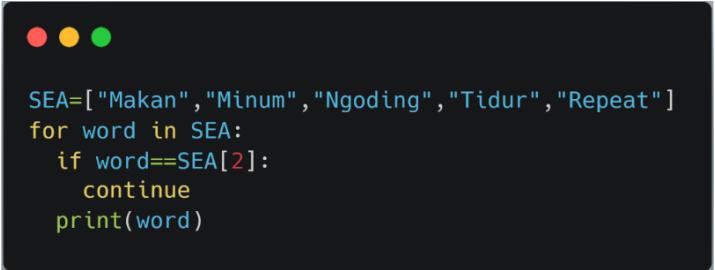
Statement lainnya tidak akan di jalankan karena perulangan sudah dihentikan oleh *break*.

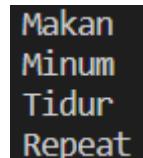
2. *Continue*

Continue digunakan untuk melanjutkan dari sebuah perulangan yang terhenti, dan kita melanjutkan ke iterasi berikutnya.

```
For ... in ....:
    Statement 1
    If kondisi:
        continue
```

Tabel 1.24 Kontrol Perulangan *Continue*

SOURCE CODE
 <pre>SEA=["Makan", "Minum", "Ngoding", "Tidur", "Repeat"] for word in SEA: if word==SEA[2]: continue print(word)</pre>
Gambar 1.55 <i>Source Code</i> Kontrol Perulangan <i>Continue</i>

OUTPUT
 <pre>Makan Minum Tidur Repeat</pre>

 Gambar 1.56 *Output* Kontrol Perulangan *Continue* |

3. Pass

Pass digunakan untuk operasi yang bersifat kosong (*null*), tidak ada yang terjadi saat pass digunakan. Biasanya *pass* digunakan saat kita ingin melakukan sebuah *statement* saat sebuah perulangan berjalan. Sintaksnya seperti berikut:

For ... in ...:
Statement I
If kondisi:
*Pass # akan
terproses nanti*

Tabel 1.25 Kontrol Perulangan *Pass*

SOURCE CODE
 <pre>SEA=["Makan" , "Minum" , "Ngoding" , "Tidur" , "Repeat"] for word in SEA: if word==SEA[2]: pass print("Ini Pass") print(word)</pre>

Gambar 1.57 *Source Code* Kontrol Perulangan *Pass*

OUTPUT
Makan Minum Ini Pass Ngoding Tidur Repeat

Gambar 1.58 *Output* Kontrol Perulangan *Pass*

4. *While Else*

Didalam perulangan *while* kita dapat menggunakan *else* yang di mana apabila kondisi *while* menjadi *false* maka *else* akan dijalankan, sintaknya sebagai berikut:

```
while kondisi :
    Statement 1
    Increment
else :
    Statement lainnya
    #kondisi while sudah bernilai false maka else
    langsung dijalankan
```

Tabel 1.26 Kontrol perulangan *While-Else*

SOURCE CODE
<pre>x=1 n=int(input("Banyaknya Perulangan : ")) while x<=n: print("Ini perulangan ke", x) x=x+1 else: print("Perulangan Telah Selesai")</pre>
<p>Gambar 1.59 Source Code Kontrol Perulangan Pass</p> <p>Banyaknya Perulangan : 3 Ini perulangan ke 1 Ini perulangan ke 2 Ini perulangan ke 3 Perulangan Telah Selesai</p>

Gambar 1.60 Output Kontrol Perulangan *While-Else*

5. *For Else*

For else biasa digunakan untuk melakukan pencarian, *for* akan melakukan perulangan sampai kondisi *if* terpenuhi kemudian program akan menjalankan *statement if* dan namun apabila dalam perulangan *if* bernilai *true*, sekali saja benar maka program *statement else* tidak akan pernah dijalankan, oleh karena itu kondisi *if* haruslah selalu salah atau *false* untuk menjalankan *statement else*. Sintaksnya sebagai berikut:

```
for ... in ... :
    If kondisi :
        statement1 "true"
        break
    #jika if bernilai false maka
    else
        dijalankan
    else :
```

Tabel 1.27 Kontrol perulangan *For-Else*

SOURCE CODE
<pre>●●● for word in "ABCDE": if word=="E": print("Tetap Semangat Belajar Lebih Giat Lagi!!!") break else: print("Selamat, Anda Lolos Ke Babak Selanjutnya")</pre>

Gambar 1.61 Source Code Kontrol Perulangan *For-Else*

SOURCE CODE
<pre>●●● for word in "ABCDE": if word=="D": print("Tetap Semangat Belajar Lebih Giat Lagi!!!") break else: print("Selamat, Anda Lolos Ke Babak Selanjutnya")</pre>

Gambar 1.62 Source Code Kontrol Perulangan *For-Else*

OUTPUT
<pre>Tetap Semangat Belajar Lebih Giat Lagi!!!</pre>
<p>Gambar 1.63 Output Kontrol Perulangan <i>For-Else</i></p>
<pre>Selamat, Anda Lolos Ke Babak Selanjutnya</pre>
<p>Gambar 1.64 Output Kontrol Perulangan <i>For-Else</i></p>

Modul 2: Struktur Data

2.1 Tujuan Praktikum

1. Memahami konsep dan penggunaan Struktur Data dalam bahasa pemrograman Python.
2. Memahami jenis-jenis Struktur Data dan tiap penggunaanya.

2.2 Alat dan Bahan

1. PC yang telah terinstall IDE Pycharm atau semacamnya

2.3 Dasar Teori

Struktur Data dalam Python memiliki beberapa tipe yang dapat digunakan, yaitu: List, Tuple, Set, Dictionary

2.3.1 List

List merupakan struktur data yang teratur atau terorganisir dan dapat diubah, list mengizinkan data rangkap atau duplikat. Dalam python list menggunakan tanda kurung siku “[]” Contoh: variable=[“data-1”, “data-2”, ” ”]

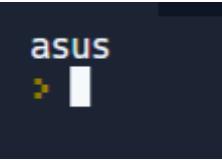
A. Mengakses data pada list

Selain itu untuk mengakses data tertentu *syntax* dapat diubah dengan mengganti angka indeksnya, berikut adalah contoh mengakses beberapa data sekaligus menggunakan indeks pada list:

Tabel 2.1 Mengakses List

SOURCE CODE
 <pre>laptop=["acer", "asus", "lenovo"] print(laptop[1]) # isi dengan indek data</pre>
OUTPUT

Gambar 2.1 Source Code Mengakses data pada List



Gambar 2.2 Output mengakses data pada List

Selain itu untuk mengakses data tertentu *syntax* dapat diubah dengan mengganti angka indeksnya, berikut adalah contoh mengakses beberapa data sekaligus menggunakan indeks pada list:

Tabel 2.2 *List-Akses*

SOURCE CODE

```
● ● ●  
laptop=["acer","asus","lenovo","dell","apple"]  
print(laptop[1:3]) #isi dengan range indeks data
```

Gambar 2.3 *Source Code List-Akses*

OUTPUT

```
['asus', 'lenovo']  
▶ []
```

Gambar 2.4 *Output List-Akses*

- a. akan mengakses atau memilih item dari posisi 1 ke 3.
- b. data pertama adalah indeks 1 dan Data pada posisi tiga tidak akan masuk.
- c. jika tidak diberi nilai awal maka range akan dimulai dari data paling pertama. Misal “print(laptop[:3])” akan membuat output mulai dari data pertama sampai ke-3.
- d. jika tidak diberi nilai akhir maka range akan dimulai data paling pertama, Misal “print(laptop[1:])” akan membuat output mulai dari data ke-2 sampai data terakhir.

B. Mengganti data pada list

Tabel 2.3 List-Ganti

SOURCE CODE
 <pre>laptop=["acer", "asus", "lenovo", "dell"] laptop[1]={"apple"} #mengganti data pada laptop print(laptop)</pre>

Gambar 2.5 Source Code List-Ganti

OUTPUT
<pre>['acer', 'apple', 'lenovo', 'dell']</pre>

Gambar 2.6 Output List-Ganti

C. Menambahkan data pada list

Untuk menambahkan data ke dalam list dapat menggunakan fungsi .append(), berikut adalah contohnya:

Tabel 2.4 List-Tambah

SOURCE CODE
 <pre>laptop=["acer", "asus", "lenovo", "dell"] laptop.append("apple") #menambah data pada list print(laptop)</pre>

OUTPUT
<pre>['acer', 'asus', 'lenovo', 'dell', 'apple']</pre>

Gambar 2.7 Source Code List-Tambah

Gambar 2.8 Output List-Tambah

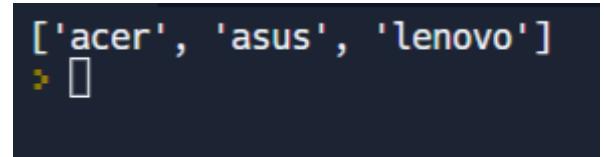
D. Menghapus data pada list

Untuk menghapus data dari dalam list dapat menggunakan fungsi `.remove()`, berikut adalah contohnya:

Tabel 2.5 *List-Hapus*

SOURCE CODE
 <pre>laptop=["acer", "asus", "lenovo", "dell"] laptop.remove("dell") print(laptop)</pre>

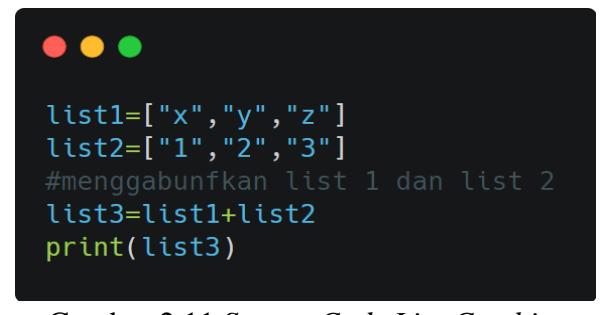
Gambar 2.9 *Source Code List-Hapus*

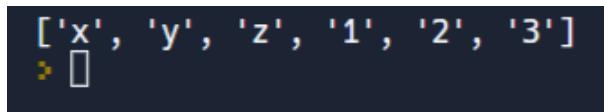
OUTPUT
 <pre>['acer', 'asus', 'lenovo'] > []</pre>

Gambar 2.10 *Output List-Hapus*

E. Menggabungkan dua list menjadi satu

Tabel 2.6 *List-Combine*

SOURCE CODE
 <pre>list1= ["x", "y", "z"] list2= ["1", "2", "3"] #menggabungkan list 1 dan list 2 list3=list1+list2 print(list3)</pre>

OUTPUT
 <pre>['x', 'y', 'z', '1', '2', '3'] > []</pre>

Gambar 2.11 *Source Code List-Combine*

Gambar 2.12 *Output List-Combine*



F. Menggurutkan data pada list

Tabel 2.7 *List-Sort* dari awal ke akhir

SOURCE CODE
laptop=["Acer","Lenovo","Dell","BenQ"] laptop.sort() #Mengurutkan data pada list print(laptop)

Gambar 2.13 *Source Code List* awal ke akir

OUTPUT
['Acer', 'BenQ', 'Dell', 'Lenovo'] :> □

Gambar 2.14 *Output List-Sort* dari awal ke akhir

Tabel 2.8 *List-Sort* dari akhir ke awal

SOURCE CODE
laptop=["Acer","BenQ","Dell","Lenovo"] laptop.sort(reverse=True) #mengurutkan data pada list print(laptop)

OUTPUT
['Lenovo', 'Dell', 'BenQ', 'Acer'] :> □

Gambar 2.15 *Source Code List-Sort* dari akhir ke awal

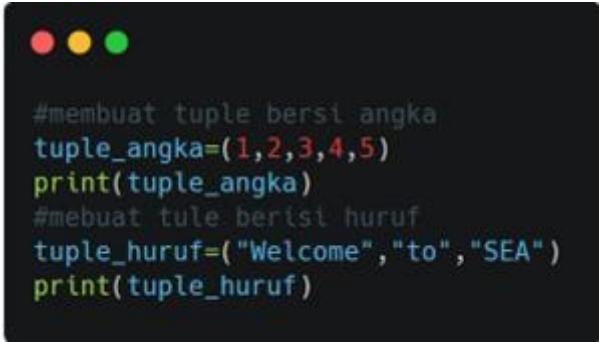
Gambar 2.16 *Output List-Sort* dari akhir ke awal

2.3.2 Tuple

Tuple merupakan struktur data yang mirip dengan *List*. *Tuple* dapat berisikan berbagai macam tipe data, dan elemen-elemennya memiliki index. Elemen-elemen pada *Tuple* dipisahkan dengan tanda koma. Dalam *Python*, *Tuple* ditulis menggunakan tanda kurung “()”.

Terdapat dua jenis *Tuple* yang dapat dibentuk secara umum. Yaitu, *Tuple* dengan satu data tipe dan *Tuple* dengan data tipe campuran. Untuk lebih memahami bagaimana membuat kedua bentuk *Tuple* tersebut, perhatikan source code *Tuple* dengan satu tipe data dan banyak data berikut:

Tabel 2.9 *Tuple-Satu Tipe*

SOURCE CODE
 <pre>#membuat tuple bersi angka tuple_angka=(1,2,3,4,5) print(tuple_angka) #mbuat tule bersi huruf tuple_huruf=("Welcome","to","SEA") print(tuple_huruf)</pre>

Gambar 2.17 Source Code *Tuple-Satu Tipe*

OUTPUT
(1, 2, 3, 4, 5) ('Welcome', 'to', 'SEA') > □

Gambar 2.18 Output *Tuple-Satu Tipe*

Tabel 2.10 *Tuple-Banyak Tipe*

SOURCE CODE



Gambar 2.19 *Source Code Tuple-Banyak Tipe*

OUTPUT
<pre>(10, 3.14, 'SEA', True, 'A')</pre>

Gambar 2.20 *Output Tuple-Banyak Tipe*

Terdapat beberapa hal yang dapat dilakukan dengan Tuple untuk mempermudah kita dalam mengolah data yang kita punya dalam sebuah program antara lain sebagai berikut:

A. Mengakses data pada tuple

Tuple bersifat immutable. Artinya adalah kita tidak dapat mengubah elemen dari Tuple setelah diinisialisasi. Untuk mengakses setiap elemen dalam Tuple, kita menggunakan index. Terdapat 2 cara dalam mengakses elemen pada Tuple, antara lain sebagai berikut:

1. Menggunakan Indeks

Tabel 2.11 *Tuple Akses Indeks*

SOURCE CODE
 <pre>#membuat tuple yang berisi angka genap angka_genap=(2,4,6,8,10,12) #mengakses tuple angka genap pada indeks pertama print("Indeks Pertama: ",angka_genap[0]) #mengakses tuple angka genap pada indeks ketiga print("Indeks Ketiga: ",angka_genap[2]) #mengakses tuple angka genap pada indeks kelima print("Indeks Kelima: ",angka_genap[4])</pre>
OUTPUT

Gambar 2.21 *Source Code Tuple Akses-Indeks*

```
Indeks Pertama: 2  
Indeks Ketiga: 6  
Indeks Kelima: 10  
:> []
```

Gambar 2.22 Output Tuple Akses-Indeks

2.

Unpacking (pembongkaran atau pemisahan)

Tabel 2.12 Tuple-Akses Unpacking

SOURCE CODE

```
#membuat tuple angka ganjil  
angka_ganjil=(1,3,5,7)  
#Unpacaking elemen pada tuple angka ganjil  
a, b, c, d = angka_ganjil  
print(a)  
print(b)  
print(c)  
print(d)|
```

Gambar 2.23 Source Code Iuple- Akses Unpacking

OUTPUT

```
1  
3  
5  
7  
:> []
```

Gambar 2.24 Output Iuple- Akses Unpacking

B. Menggabungkan 2 tuple (*Conccatenation*)

Tuple bersifat immutable. Artinya adalah kita tidak dapat mengubah elemen dari Tuple setelah diinisialisasikan. Untuk mengakses setiap elemen dalam Tuple, kita menggunakan index. Terdapat 2 cara dalam mengakses elemen pada Tuple, antara lain sebagai berikut:

Tabel 2.13 Tuple-Concatenation

SOURCE CODE

```
tuple1=(3,2,1)
tuple2=("Welcome","To","SEA")
#concatenation tuple 1 dan 2
tuple3=tuple1+tuple2
#menampilkan ketiga tuple dan lihat perbedaannya
print(tuple1)
print(tuple2)
print(tuple3)
```

Gambar 2.25 Source Code Tuple-Concatenation

OUTPUT

```
(3, 2, 1)
('Welcome', 'To', 'SEA')
(3, 2, 1, 'Welcome', 'To', 'SEA')
> █
```

Gambar 2.26 Output Tuple-Concatenation

C. Mengiris tuple (*Slicing*)

Mengiris Tuple berarti mengambil bagian kecil dari sebuah tuple. Misalkan kita mempunyai tuple dengan panjang elemen 5, dan kita mengiris tuple tersebut dari index 1 sampai 3 dan mendapatkan tuple baru atau sub-tuple. Untuk lebih memahaminya, perhatikan contoh berikut:

Tabel 2.14 Tuple-Slicing

SOURCE CODE

```
tuple_angka=(1,2,3,4,5,6)
#slicing tuple_angka menghapus angka pertama
print(tuple_angka[1:])
#slicing tuple_angka menampilkan 3 angka pertama
print(tuple_angka[:3])
#slicing tuple_angka menghapus angka terakhir
print(tuple_angka[:-1])
```

Gambar 2.27 Source Code Tuple-Slicing

OUTPUT

```
(2, 3, 4, 5, 6)
(1, 2, 3)
(1, 2, 3, 4, 5)
> □
```

Gambar 2.28 Output *Tuple-Slicing*

D. Menghapus tuple (*Delete*)

Seperti yang sudah kita ketahui sebelumnya bahwa tuple adalah objek yang immutable. Maka kita tidak dapat mengubah atau menghapus elemen dari tuple. Akan tetapi kita dapat menghapus tuple secara keseluruhan dengan key-word del. Perhatikan contoh berikut:

Tabel 2.15 *Tuple-Delete*

SOURCE CODE

```
#menghapus tuple
tuple0=("Good", "Morning", "SEA")
print(tuple0)
del tuple0
print("Berhasil dihapus")
```

Gambar 2.29 *Source Code Tuple-Delete*

OUTPUT

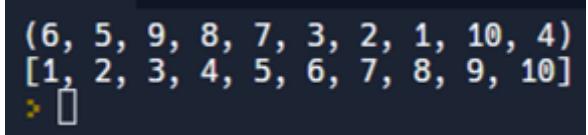
```
('Good', 'Morning', 'SEA')
Berhasil dihapus
> □
```

Gambar 2.30 *Output Tuple-Delete*

E. Mengurutkan elemen tuple(*Sorting*)

Seperti Elemen-elemen pada tuple juga dapat diurutkan secara mudah dengan built-in method yang dimiliki oleh tuple dalam Bahasa Python yaitu sorted(). Fungsi ini akan menghasilkan sebuah list. Perhatikan contoh berikut:

Tabel 2.16 *Tuple-Sort*

SOURCE CODE
 <pre>tuple=(6,5,9,8,7,3,2,1,10,4) #sebelum diurutkan print(tuple) #sesudah diurutkan tuple=sorted(tuple) print(tuple)</pre>
<p>Gambar 2.31 <i>Source Code Tuple-Sort</i></p>
OUTPUT
 <pre>(6, 5, 9, 8, 7, 3, 2, 1, 10, 4) [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] > █</pre>
<p>Gambar 2.32 <i>Output Tuple-Sort</i></p>

2.3.3 Set

Set merupakan struktur data yang tidak teratur atau terorganisir dan tidak memiliki indeks sehingga membuatnya tidak dapat di akses dengan menggunakan indeks. *Set* tidak mengizinkan anggota rangkap atau duplikat. Dalam Python *Set* biasanya digunakan untuk melakukan operasi himpunan matematika seperti gabungan, irisan, selisih, dll. *Set* ditulis menggunakan tanda kurung kurawal “{ }”.

Selain itu data di dalam sebuah *Set* tidak dapat diubah setelah *Set* dibuat, tetapi dapat ditambahkan data atau menghapus data dalam sebuah *Set*.

A. Menambahkan data dalam Set

Untuk menambahkan data ke dalam *list* dapat menggunakan fungsi *add()* dan jika ingin menambahkan beberapa data sekaligus dapat menggunakan fungsi *update()*.

a. Fungsi *add()*

Tabel 2.17 *Set-Add*

SOURCE CODE
 <pre>buah={"apel","jeruk","pisang"} buah.add("mangga") #menambahkan data pada set print(buah)</pre>
Gambar 2.33 <i>Source Code Set-Add</i>
OUTPUT
<pre>{'mangga', 'jeruk', 'pisang', 'apel'}</pre>
Gambar 2.34 <i>Output Set-Add</i>

b. Fungsi *update()*

Tabel 2.18 *Set-Update*

SOURCE CODE
 <pre>buah={"apel","jeruk","pisang"} buah.update(["mangga","anggur"]) #menambahkan data pada set print(buah)</pre>

Gambar 2.35 *Source Code Set-Update*

OUTPUT

```
{'apel', 'mangga', 'jeruk', 'anggur', 'pisang'}
```

Gambar 2.36 Output Set-Update

B. Menghapus data dalam Set

Untuk menghapus data dalam set dapat menggunakan beberapa cara yaitu *remove*, *discard*, dan *clear*.

Tabel 2.19 Set-Delete

SOURCE CODE

```
● ○ ●

buah={"apel", "jeruk", "pisang", "mangga"}
buah.remove("mangga") #menghapus 1 data pada set
print(buah)

buah.discard("jeruk") #menghapus 1 data pada set
print(buah)

buah.clear() #menghapus semua data pada set
print(buah)
```

Gambar 2.37 Source Code Set-Delete

OUTPUT

```
{'jeruk', 'pisang', 'apel'}
{'pisang', 'apel'}
set()
```

Gambar 2.38 Output Set-Delete

C. Menggabungkan dua Set menjadi satu

Tabel 2.20 *Set-Combine*

SOURCE CODE

```
● ● ●  
set1={"a", "b", "c"}  
set2={1,2,3}  
  
set1.update(set2) #menggabungkan 2 set  
print(set1)
```

Gambar 2.39 *Source Code Set-Combine*

OUTPUT

```
{1, 2, 3, 'b', 'c', 'a'}  
▼
```

Gambar 2.40 *Output Set-Combine*

2.3.4 Dictionary

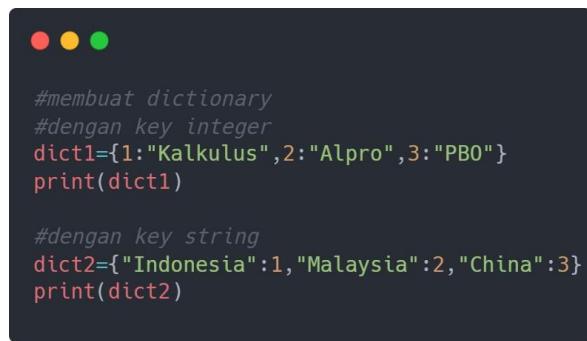
Dictionary merupakan struktur data yang lebih kompleks dari yang lainnya (*list*, *tuple*, *set*). Yang membedakan *dictionary* dengan struktur data lain adalah elemen dalam *dictionary* yang merupakan pasangan *key* dan *value*. Sedangkan untuk struktur data lainnya hanyalah nilai tunggal.

Dictionary jika diartikan dalam Bahasa Indonesia artinya kamus. Konsepnya di pemrograman juga sama dengan ketika kita menggunakan kamus di dunia nyata, kita akan mencari kata dengan sebuah kata kunci yaitu alfabet yang terletak di lembaran kamus. Pada Python kita menemukan *value* dari elemen *dictionary* dengan berpatokan pada *key* yang tersambung ke *value* yang kita cari.

Key pada *dictionary* dapat berupa tipe data apapun. Bisa menggunakan integer ataupun string. Perhatikan contoh pembuatan sebuah *dictionary* berikut ini:

Tabel 2.21 *Dictionary-Deklarasi 1*

SOURCE CODE



```
#membuat dictionary
#dengan key integer
dict1={1:"Kalkulus",2:"Alpro",3:"PBO"}
print(dict1)

#dengan key string
dict2={"Indonesia":1,"Malaysia":2,"China":3}
print(dict2)
```

Gambar 2.41 *Source Code Dictionary*-Deklarasi 1

OUTPUT

```
{1: 'Kalkulus', 2: 'Alpro', 3: 'PBO'}
{'Indonesia': 1, 'Malaysia': 2, 'China': 3}
> |
```

Gambar 2.42 *Output Dictionary*-Deklarasi 1

Selain dengan cara diatas. Ada satu cara lagi untuk mendeklarasikan sebuah *dictionary*. Yaitu dengan menggunakan *method dict()*. Perhatikan contoh berikut:

Tabel 2.22 *Dictionary*-Deklarasi 2

SOURCE CODE



```
#membuat dictionary
#dengan dict()
bioDict = dict(
    {"Name": "Aditya", "Age": 20}
)
print(bioDict)
```

Gambar 2.43 Source Code Dictionary-Deklarasi 2

OUTPUT

```
{'Name': 'Aditya', 'Age': 20}
```

Gambar 2.44 Output Dictionary-Deklarasi 2

Terdapat beberapa hal yang dapat dilakukan dengan *dictionary* untuk mempermudah kita dalam mengelola data kita dalam program, antara lain sebagai berikut:

A. Menambahkan elemen (*Add*) dan *update* elemen

Dictionary bersifat *mutable* artinya elemen di dalamnya dapat diubah atau diperbaharui setelah dideklarasikan. Perhatikan contoh berikut:

Tabel 2.23 *Dictionary*- Menambah dan Update Elemen

SOURCE CODE

```
● ● ●  
#mendeklarasikan dictionary  
bioDict = {}  
print(bioDict)  
#menambahkan elemen baru  
#syntax: dict_name[key] = value  
bioDict["Name"] = "Aditya"  
bioDict["Age"] = "20"  
bioDict["Work"] = "Developer"  
print(bioDict)  
#update elemen  
#syntax: dict_name[key] = new_value  
bioDict["Work"] = "Student"  
print(bioDict)
```

Gambar 2.45 Source Code Dictionary-tambah dan Update

OUTPUT

```
{}  
{'Name': 'Aditya', 'Age': '20', 'Work': 'Developer'}  
{'Name': 'Aditya', 'Age': '20', 'Work': 'Student'}  
▶ █
```

Gambar 2.46 Dictionary-tambah dan Update

B. Mengakses elemen

Untuk mengakses suatu nilai dari elemen dalam *dictionary* kita tidak lagi menggunakan *index* melainkan menggunakan *key*. Perhatikan contoh berikut:

Tabel 2.24 Dictionary- Akses Key

SOURCE CODE



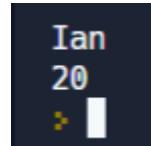
```

● ● ●

bioDict = dict({"Name":"Ian", "Age":20})
#mengakses nilai dari elemen
print(bioDict["Name"])
print(bioDict["Age"])

```

OUTPUT



```

Ian
20
> 

```

Gambar 2.47 *Source Code Dictionary- Akses Elemen*

Gambar 2.48 *Output Dictionary- Akses Elemen*

Selain dengan cara diatas. Mengakses nilai dari elemen dalam *dictionary* dapat dilakukan juga dengan *method get()*. Perhatikan contoh berikut:

Tabel 2.25 *Dictionary- Akses Get*

SOURCE CODE



```

● ● ●

dict1 = {1:"Kalkulus",2:"Alpro",3:"PBO"}
#mengakses nilai dari elemen
print(dict1.get(1))
print(dict1.get(2))
print(dict1.get(3))

```

OUTPUT

Gambar 2.49 *Source Code Dictionary- Akses Get*

OUTPUT

```
Kalkulus  
Alpro  
PBO  
▶
```

Gambar 2.50 *Output Dictionary- Akses Get*

C. Menghapus elemen (*Delete*)

Ada beberapa pilihan cara untuk menghapus sebuah elemen dari *dictionary*, antara lain adalah sebagai berikut:

a. Menggunakan *del keyword*

Kita dapat menghapus sebuah elemen dari *dictionary* dengan mudah menggunakan *keyword del*. Perhatikan contoh berikut:

Tabel 2.26 *Dictionary-Delete-del*

SOURCE CODE

```
#dictionary  
dict1 = {1:"Kalkulus",2:"Alpro",3:"PBO"}  
print(dict1)  
#menghapus elemen  
del dict1[1]  
print(dict1)  
#menghapus elemen  
del dict1[2]  
print(dict1)
```

Gambar 2.51 *Source Code Dictionary-Delete-del*

OUTPUT

```
{1: 'Kalkulus', 2: 'Alpro', 3: 'PBO'}  
{2: 'Alpro', 3: 'PBO'}  
{3: 'PBO'}  
▶
```

Gambar 2.52 *Output Dictionary-Delete-del*

b. Menggunakan method *pop()*

Method *pop()* digunakan untuk menghapus elemen dari *dictionary* sesuai dengan *key*. Perhatikan contoh berikut:

Tabel 2.27 *Dictionary- Delete-pop*

SOURCE CODE
<pre>#dictionary dict1 = {1:"Kalkulus",2:"Alpro",3:"PBO"} print(dict1) #menghapus elemen dict1.pop(1) print(dict1) #menghapus elemen dict1.pop(2) print(dict1)</pre>
<pre>{1: 'Kalkulus', 2: 'Alpro', 3: 'PBO'} {2: 'Alpro', 3: 'PBO'} {3: 'PBO'}</pre>

Gambar 2.53 *Code Dictionary- Delete-pop*

Gambar 2.54 *Output Dictionary- Delete-pop*

c. Menggunakan method *popitem()*

Berbeda dengan *pop()*, method *popitem()* akan menghapus elemen paling kanan atau elemen terakhir. Perhatikan contoh berikut:

Tabel 2.28 *Dictionary- Delete-popitem*

SOURCE CODE

```
#dictionary  
dict1 = {1:"Kalkulus",2:"Alpro",3:"PBO"}  
print(dict1)  
#menghapus elemen  
dict1.popitem()  
dict1.popitem()  
print(dict1)
```

Gambar 2.55 Source Code Dictionary- Delete-popitem

OUTPUT

```
{1: 'Kalkulus', 2: 'Alpro', 3: 'PBO'}  
{1: 'Kalkulus'}  
> |
```

Gambar 2.56 Output Dictionary- Delete-popitem

d. Menggunakan method clear()

Method *clear()* akan menghapus seluruh elemen dari sebuah *dictionary*. Perhatikan contoh berikut:

Tabel 2.29 Dictionary- Delete-clear

```
#dictionary  
dict1 = {1:"Kalkulus",2:"Alpro",3:"PBO"}  
print(dict1)  
#menghapus elemen  
dict1.clear()  
print(dict1)
```

Gambar 2.57 Source Code Dictionary- Delete-clear

OUTPUT

```
{1: 'Kalkulus', 2: 'Alpro', 3: 'PBO'}  
{}  
> |
```

Gambar 2.58 Output Dictionary- Delete-clear

Modul 3: Fungsi, Class dan Inheritance

3.1 Tujuan Praktikum

1. Memahami dan mampu mengimplementasikan Fungsi dan Class
2. Memahami dan mampu implementasi Inheritance
3. Memahami konsep OOP dan dapat menimplementasikannya dengan baik

3.2 Alat dan Bahan

1. PC yang terinstall Python 3
2. PC yang terinstall IDE PyCharm / Visual Studio Code

3.3 Dasar Teori

3.3.1 Fungsi

Fungsi adalah sebuah blok kode terorganisir yang dapat digunakan kembali atau reusable untuk melakukan sebuah aksi / tindakan. Fungsi terpisah dalam blok sendiri yang berfungsi sebagai *sub-program* yang merupakan sebuah *program* kecil untuk memproses sebagian dari pekerjaan *program* utama. Deklarasi kode dari sebuah fungsi adalah sebagai berikut:

```
def nama_fungsi(parameter):
    #statement kode
```

Gambar 3.1 Deklarasi Sebuah Fungsi

Untuk pemanggilan fungsi, bentuk umumnya adalah sebagai berikut:

```
nama_fungsi(parameter)
```

Gambar 3.2 Pemanggilan Fungsi

Berdasarkan nilai balik (*return*), fungsi dibedakan menjadi 2, yaitu:

A. Fungsi dengan Return

Fungsi dengan *return* statement yaitu fungsi yang dapat mengembalikan nilai saat dijalankan. Fungsi dengan *return* diperlukan

ketika kita menginginkan hasil sebuah *value* (nilai) dan menggunakan *value* tersebut pada proses yang lain. Kita bisa mengembalikan nilai dengan menggunakan perintah *return* lalu diikuti dengan nilai atau variabel yang akan dikembalikan.

Tabel 3.1 Fungsi - dengan *Return*

SOURCE CODE
<pre>● ○ ● def kali(a,b): return a*b #contoh penggunaan y = kali(4,5) print("hasilnya adalah", y)</pre>

Gambar 3.3 Source Code Fungsi dengan *Return*

OUTPUT
<pre>hasilnya adalah 20 > █</pre>

Gambar 3.4 Output Fungsi dengan *Return*

Pada contoh di atas, kita membuat fungsi *kali()* yang berfungsi untuk melakukan perkalian dua bilangan. Fungsi *kali()* memiliki dua parameter, *a* dan *b*. Parameter *a* dan *b* berperan sebagai penampung nilai bilangan yang akan dikalikan. Saat fungsi *kali()* dipanggil, fungsi akan me-*return* hasil perkalian dari parameter *a* dan *b*. Karena fungsi *kali()* melakukan *return* nilai, maka kita dapat menampung nilai yang dikembalikan oleh fungsi *kali()* ke dalam suatu variabel.

B. Fungsi tanpa *Return*

Fungsi tanpa *return* adalah fungsi yang tidak mengembalikan nilai saat dipanggil. Untuk membuat fungsi tanpa *return*, kita tidak perlu menyertakan perintah *return* di bagian akhir fungsi.

Tabel 3.2 Fungsi - tanpa *Return*

SOURCE CODE



```
def kali(a,b):
    print("hasilnya adalah", a*b)

#contoh penggunaan
kali(4,5)
```

Gambar 3.5 *Code* Fungsi tanpa *Return*



OUTPUT

```
hasilnya adalah 20
```

Gambar 3.6 *Output* Fungsi tanpa *Return*

Pada contoh di atas, kita membuat fungsi kali() yang berfungsi untuk melakukan perkalian dua bilangan. Fungsi kali() memiliki dua parameter, a dan b. Parameter a dan b berperan sebagai penampung nilai bilangan yang akan dikalikan. Saat fungsi kali() dipanggil, fungsi akan melakukan perkalian dan menampilkan hasil tanpa me-*return* suatu nilai ke program utama. Karena fungsi kali() tidak melakukan *return* nilai, fungsi kali() tidak bisa berinteraksi dengan program utama. Sehingga, kita harus menyelesaikan semua aksi / tindakan di dalam fungsi tersebut. Pada materi fungsi ini, pemrograman python memiliki aturan dalam mendeklarasikan suatu variabel. Diantaranya adalah sebagai berikut:

a. Variabel Lokal

Dalam pemrograman python, variabel lokal merupakan variabel yang dideklarasikan di dalam sebuah fungsi. variabel lokal hanya dapat diakses pada fungsi tempat variabel tersebut dibuat. Sehingga, jika kita menggunakan variabel tersebut di luar fungsinya, akan terjadi *error* pada program.

Tabel 3.3 Fungsi – Variabel Lokal

SOURCE CODE

```
def sebut_nama():
    #deklarasi variabel nama
    nama = "Ilham"
    print("Akses di dalam fungsi :")
    print(nama)

#memanggil fungsi sebut_nama
sebut_nama()

#akses variabel nama dari luar fungsi
print("Akses di luar fungsi :")
print(nama)
```

Gambar 3.7 *Source Code* Fungsi – Variabel Lokal

OUTPUT

```
Akses di dalam fungsi :
Ilham
Akses di luar fungsi :
Traceback (most recent call last):
  File "main.py", line 12, in <module>
    print(nama)
NameError: name 'nama' is not defined
> █
```

Gambar 3.8 *Output* Fungsi – Variabel Lokal

b. Variabel Global

Variabel Global adalah variabel yang dideklarasikan di luar fungsi atau dalam lingkup global. Karena variabel ini bersifat global, variabel global dapat diakses di luar ataupun di dalam suatu fungsi.

Tabel 3.4 Fungsi – Variabel Global

SOURCE CODE

```
kata = "aku variabel global"
def print_kata():
    #akses variabel kata dari dalam fungsi
    print("Akses di dalam fungsi :",kata)

#memanggil fungsi print_kata
print_kata()

#akses variabel kata dari luar fungsi
print("Akses di luar fungsi :",kata)
```

Gambar 3.9 *Source Code* Fungsi – Variabel Global

OUTPUT

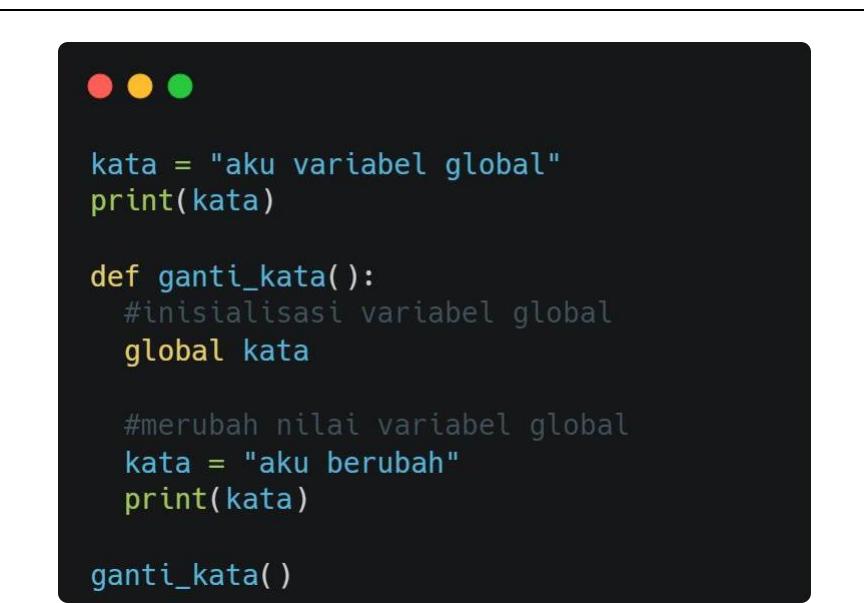
```
Akses di dalam fungsi : aku variabel global
Akses di luar fungsi : aku variabel global
> █
```

Gambar 3.10 *Output* Fungsi – Variabel Global

Jika kita ingin mengubah nilai dari variabel global dari dalam suatu fungsi, variabel tersebut harus kita inisialisasi sebagai variabel global. Kita bisa menggunakan perintah *global* diikuti dengan nama variabel untuk inisialisasi variabel tersebut sebagai variabel global.

Tabel 3.5 Fungsi – Merubah Nilai Variabel Global

SOURCE CODE



```

● ○ ●

kata = "aku variabel global"
print(kata)

def ganti_kata():
    #inisialisasi variabel global
    global kata

    #merubah nilai variabel global
    kata = "aku berubah"
    print(kata)

ganti_kata()

```

Gambar 3.11 *Source Code* Fungsi – Merubah Nilai Variabel Global

OUTPUT
akу variabel global akу berubah > █

Gambar 3.12 *Output* Fungsi – Merubah Nilai Variabel Global

3.3.2 Class

Pada bahasa pemrograman Python, kelas didefinisikan dengan *keyword* “*class*”. Objek tertentu dapat dipresentasikan dengan *class*, sehingga akan membantu proses dalam menyelesaikan masalah-masalah kompleks dengan menyebutkan ciri fisik dari suatu masalah. *Default*-nya kelas pada Python mempunyai *Access Specifier* berbentuk *Public*.

Pada Python, sebuah *class* juga dapat memiliki variabel kelas. Variabel kelas adalah variabel yang dibagi / dapat diakses oleh semua *instance* (turunan) kelas. Variabel kelas harus didefinisikan di dalam kelas dan juga harus didefinisikan di luar fungsi-fungsi yang ada pada kelas tersebut.

Dengan mengimplementasikan *class*, kita dapat membuat suatu objek. Objek adalah *instance* atau representasi dari sebuah *class*. Jika *class* adalah sebuah cetakan, maka objek adalah hasil dari cetakan tersebut. Berikut adalah hal-hal lain yang perlu diketahui untuk mendukung pengimplementasian *class*:

A. Metode Konstruktor

Metode konstruktor adalah sebuah metode yang akan dipanggil ketika kita menginisialisasikan objek dari suatu kelas. Pada Python, Metode konstruktor didefinisikan sebagai fungsi “`_init_()`”. Fungsi tersebut akan selalu berjalan ketika program memanggil suatu *class*. Fungsi “`_init_()`” berfungsi untuk menetapkan nilai ke object properties, atau operasi lain yang perlu dilakukan saat objek dibuat. Berikut adalah contoh dari penggunaan metode konstruktor:

Tabel 3.6 Class – Contoh 1

The screenshot shows a terminal window with a dark background and light-colored text. At the top, it says "SOURCE CODE". Below that is the Python code:

```
class Mahasiswa:
    univ = "Tel-U" #contoh variabel kelas

    def __init__(self,nama,jurusan): #metode konstruktor
        self.nama = nama
        self.jurusan = jurusan

#contoh pembuatan objek dari class Mahasiswa
mahasiswa1 = Mahasiswa("Budi","Teknik Komputer")

#print nama dari objek mahasiswa1
print(mahasiswa1.nama)

#print jurusan dari objek mahasiswa1
print(mahasiswa1.jurusan)

#print variabel kelas
print(mahasiswa1.univ)
```

Gambar 3.13 Source Code Class – Contoh 1

The screenshot shows a terminal window with a dark background and light-colored text. At the top, it says "OUTPUT". Below that is the output of the printed variables:

```
Budi
Teknik Komputer
Tel-U
> |
```

Gambar 3.14 Output Class – Contoh 1

Pada contoh diatas, “`self`” adalah parameter referensi ke instansi *class* saat ini, dan digunakan untuk mengakses variabel yang termasuk dalam *class*. Parameter tersebut tidak harus dinamai “`self`”, user bebas menamai parameter tersebut dengan nama apapun, tetapi parameter itu harus menjadi parameter pertama dari setiap fungsi pada *class*.

Kita dapat memanggil *class* dengan cara mendeklarasikan terlebih dahulu sebuah *object*, seperti yang dicontohkan diatas (`mahasiswa1`). Setelah *object* dibuat, kita dapat menampilkan isi dari *class* dengan perintah “`print(nombreObject.nombreMethode)`”.

Tabel 3.7 Class – Contoh 2

SOURCE CODE
<pre>● ● ●\n\nclass Persegi:\n def __init__(self,sisi):\n self.sisi = sisi\n\n def luas(self):\n return self.sisi * self.sisi\n\n def keliling(self):\n return self.sisi * 4\n\n#contoh penggunaan\nobj = Persegi(5) # membuat objek\nprint(obj.luas()) # memanggil metode luas\nprint(obj.keliling()) # memanggil metode keliling</pre>

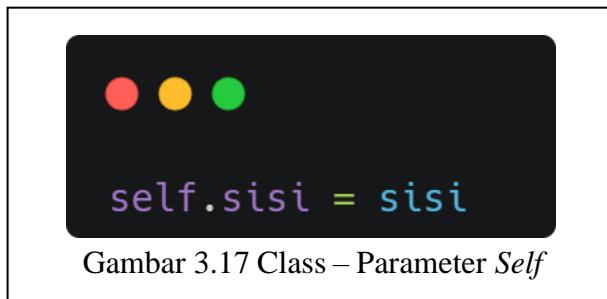
Gambar 3.15 Source Code Class – Contoh 2

OUTPUT
<pre>25\n20\n> █</pre>

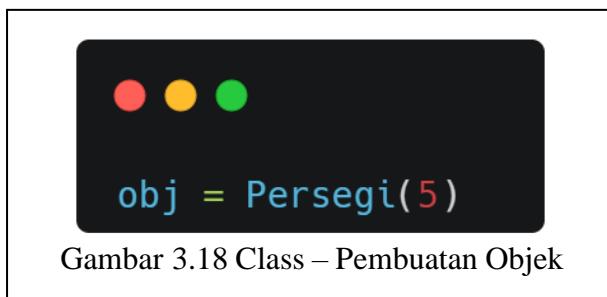
Gambar 3.16 Output Class – Contoh 2

Kelas segiempat yang didefinisikan diatas memiliki satu atribut, yaitu sisi. Kelas tersebut juga memiliki tiga metode, yaitu __init__(), luas() dan keliling(). Metode __init__() berfungsi sebagai konstruktor saat melakukan inisialisasi / pengisian nilai awal yang diperlukan. Terdapat dua parameter pada metode __init__(), yaitu self dan sisi. Parameter pertama, self, wajib disertakan untuk setiap metode yang didefinisikan didalam kelas dan tidak perlu disebutkan pada saat pemanggilan.

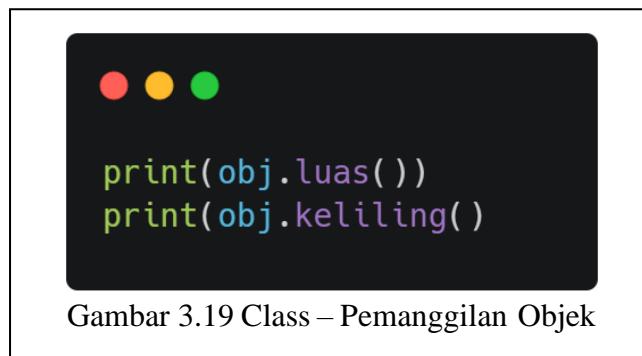
Untuk mendefinisikan atau mengakses atribut suatu kelas, kita juga perlu menggunakan kata kunci *self*. Sebagai contoh, pada kode di atas kita mendefinisikan satu atribut di dalam kelas Persegi, yaitu sisi diambil dari parameter sisi. Dengan demikian, penulisan kodennya dilakukan dengan cara sebagai berikut:



Cara membuat objek dari *class* adalah sebagai berikut:



Baris kode di atas akan membuat objek dari kelas Persegi. Nilai 5 akan disalin ke atribut sisi yang ada pada *class*. Selanjutnya, kita dapat memanggil metode-metode yang didefinisikan di dalam kelas Persegi melalui referensi objeknya, seperti dalam contoh berikut:



3.3.3 Inheritance

Inheritance adalah sebuah proses dimana sebuah class mengambil semua properti dan semua metode dari kelas lain. Syarat penggunaan konsep *inheritance* pada Python adalah terdapat *superclass* (kelas induk) dan *subclass* (kelas turunan). Suatu kelas dapat diturunkan dari kelas yang sudah didefinisikan sebelumnya. Jika kelas B diturunkan dari kelas A maka B mewarisi sifat-sifat (daftar atribut dan metode) yang dimiliki oleh kelas A. Maka setiap metode dan variabel yang terdapat di superclass dapat diakses melalui subclass tergantung dari akses metodenya.

A. Syntax

Dalam Python, kelas turunan didefinisikan menggunakan bentuk umum berikut:

```
class NamaKelasTurunan(NamaKelasInduk):  
    #Badan Kelas
```

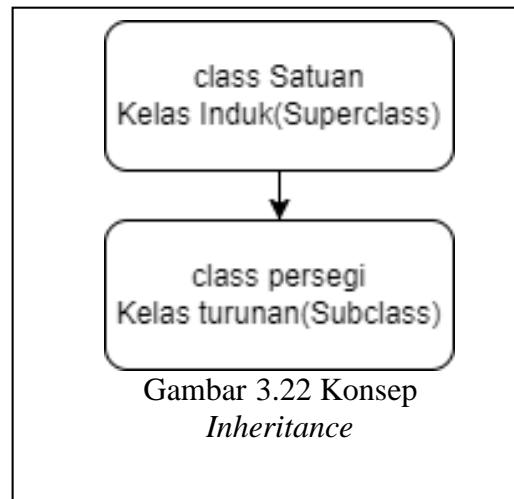
Gambar 3.20 Inheritance - Syntax

Jika nama kelas induk tidak disertakan, maka kelas tersebut secara otomatis akan diturunkan dari *class object*, yaitu kelas moyang dari semua kelas yang terdapat di dalam Python. Supaya lebih memahami mengenai *Inheritance* kita akan membuat sebuah kasus sederhana.

```
#Superclass  
class Satuan:  
    def __init__(self):  
        self.sisi = 6  
  
#Subclass  
class persegi(Satuan):  
    def __init__(self):  
        super().__init__()  
    def getluas(self):  
        self.luas = int(self.sisi) * int(self.sisi)  
        print("Luas Persegi = ",self.luas)
```

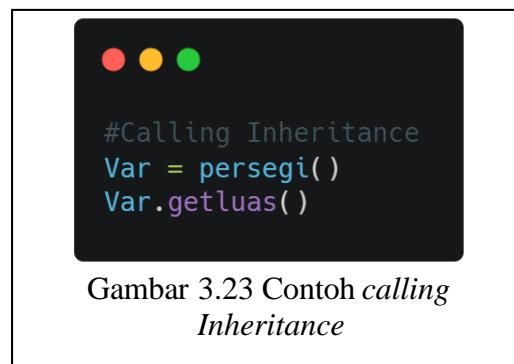
Gambar 3.21 Contoh *Inheritance*

Pada potongan *sourcecode* diatas, dibuat sebuah kelas “Satuan” yang akan digunakan sebagai Kelas Induk (*Superclass*). Dan kelas “persegi” yang akan menjadi Kelas turunan (*Subclass*) yang akan mewarisi variabel dan metode dari kelas induk dengan *Access Modifier Public* yang artinya semua Variabel dan Metode yang ada pada Kelas “Satuan” dengan *Access Modifier Public* dapat diturunkan ke Kelas “persegi”.



B. Memanggil Inheritance

Untuk pemanggilan *inheritance*, bentuk umumnya seperti berikut:



Pada metode utama, kita membuat sebuah objek dari Kelas Turunan “persegi” yaitu “Var”. maka secara otomatis Variabel dan Metode dengan *Access Modifier Public* dan *Protected* yang terdapat pada kelas utama “Satuan” dapat diakses hanya dengan membuat objek dari kelas turunannya, sehingga variabel sisi dapat diakses di metode utama begitu juga getluas().

Tabel 3.8 *Inheritance*

SOURCE CODE

```
#Superclass
class Satuan:
    def __init__(self):
        self.sisi = 6

#Subclass
class persegi(Satuan):
    def __init__(self):
        super().__init__()
    def getluas(self):
        self.luas = int(self.sisi) * int(self.sisi)
        print("Luas Persegi =", self.luas)

#CallingInheritance
Var = persegi()
Var.getluas()
```

Gambar 3.24 Sourcecode Inheritance

OUTPUT

```
Luas Persegi = 36
```

Gambar 3.25 Output Inheritance

C. Access Modifier Inheritance

Ketika menurunkan sebuah kelas, kita harus memberi sebuah *Access Modifier* untuk kelas induknya, *Access Modifier* tersebut akan mempengaruhi jenis akses pada variabel dan metode yang diturunkan. Ada 3 jenis *Inheritance* berdasarkan *Access Modifier* pada Python:

- Public** : Semua metode yang didefinisikan di dalam kelas akan bersifat *public*, sehingga bagian program diluar kelas diizinkan untuk mengakses metode tersebut secara langsung.

```
def namafungsi(self):
    #metode
```

Gambar 3.26 Access Modifier Public

- Protected** : Metode yang hanya dapat dikenal di lingkungan kelas turunannya. Dengan menyertakan tanda *underscore* yang ditulis satu kali (_) didepan nama metode atau propertiinya.

```
● ● ●  
def _namafungsi(self):  
    #metode
```

Gambar 3.27 Access Modifier Protected

- c. **Private** : Metode yang tidak dapat diakses dari luar (seakan-akan bersifat private). Dengan menyertakan tanda *underscore* yang ditulis dua kali (_ _) di depan nama metode atau propertinya.

```
● ● ●  
def __namafungsi(self):  
    #metode
```

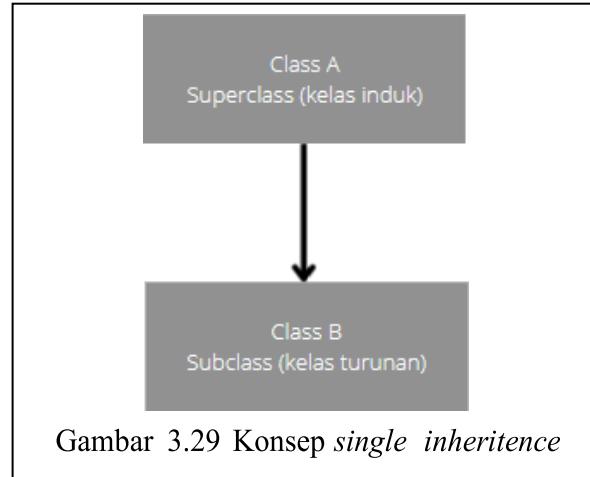
Gambar 3.28 Access Modifier Private

D. Type of Inheritance

Inheritance dapat didefinisikan sebagai mekanisme “pewarisan” properti dari kelas utama nya ke kelas anak. Tipe dari *inheritance* bergantung kepada jumlah dari kelas anak dan kelas orang tua (kelas utama) yang terlibat dari program, ada empat tipe *inheritance* pada bahasa pemrograman python:

a. Single Inheritance (pewarisan tunggal)

Single Inheritance memugkinkan kelas turunan untuk mewarisi properti dari satu kelas orang tua. Sehingga memungkinkan penggunaan kembali kode dan penambahan fitur baru ke kode yang ada.



~Berikut adalah contoh pengimplementasian *Single Inheritance*:

Tabel 3.9 Contoh *Single Inheritance*

SOURCECODE
<pre> ● ● ● class parent: def func1(self): print("ini fungsi ini ada pada kelas utama") class anak(parent): def func2(self): print("fungsi ini ada pada kelas turunan") object = anak() object.func1() object.func2() </pre>

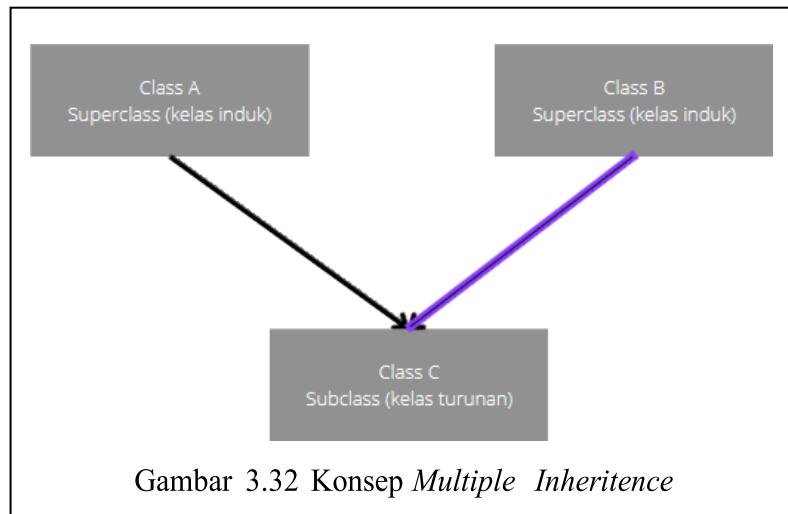
Gambar 3.30 Source Code *single Inheritance*

OUTPUT
<pre> ini fungsi ini ada pada kelas utama fungsi ini ada pada kelas turunan </pre>

Gambar 3.31 Output *single Inheritance*

b. Multiple Inheritance (Pewarisan Berganda)

Ketika sebuah kelas dapat diturunkan dari lebih dari satu kelas utama. Dalam *multiple inheritance*, semua fitur dari kelas utama dapat diwarisi kedalam kelas turunannya.



Berikut adalah contoh pengimplementasian *Multiple Inheritance*:

Tabel 3.10 Contoh *Multiple Inheritance*

SOURCECODE
<pre> ● ● ● class ibu: nama_ibu = "" def ibu(self): print(self.nama_ibu) class ayah: nama_ayah = "" def ayah(self): print(self.nama_ayah) class anak(ibu,ayah): def orang_tua(self): print("Ayah : ", self.nama_ayah) print("Ibu : ", self.nama_ibu) s1=anak() s1.nama_ayah = "Johny" s1.nama_ibu = "Jihna" s1.orang_tua() </pre>

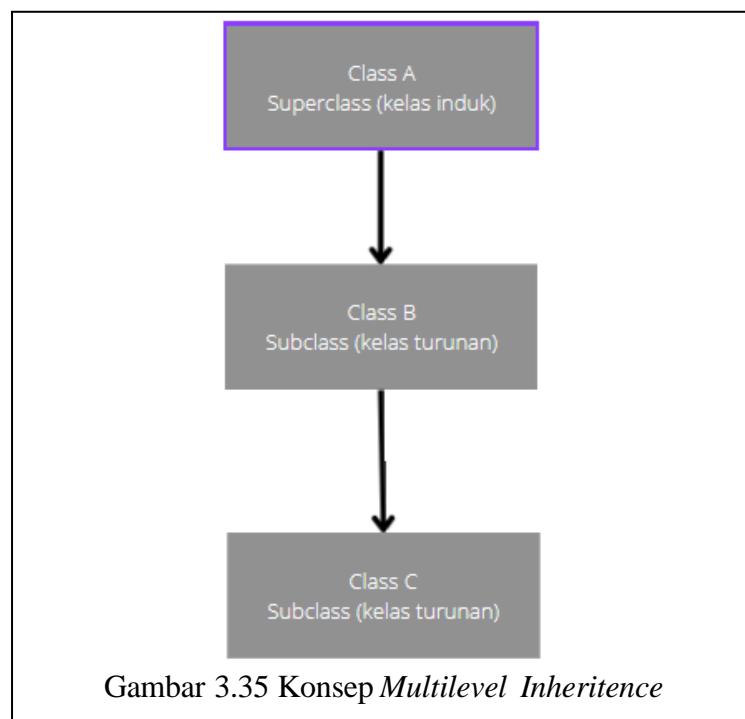
Gambar 3.33 Source Code *Multiple Inheritance*

OUTPUT
<pre> Ayah : Johny Ibu : Jihna </pre>

Gambar 3.34 Output *Multiple Inheritance*

c. Multilevel Inheritance (Pewarisan Bertingkat)

Dalam *Multilevel Inheritance*, fitur dari kelas utama dan kelas turunan akan diwarisan ke dalam kelas turunan selanjutnya yang baru.



Berikut adalah contoh pengimplementasian *Multilevel Inheritance*:

Tabel 3.11 Contoh *Multilevel Inheritance*

SOURCECODE

```

● ● ●

class kakek:
    def __init__(self, nama_kakek):
        self.nama_kakek = nama_kakek

class ayah(kakek):
    def __init__(self, nama_ayah, nama_kakek):
        self.nama_ayah = nama_ayah
        kakek.__init__(self, nama_kakek)

class anak(ayah):
    def __init__(self, nama_anak, nama_ayah, nama_kakek):
        self.nama_anak = nama_anak
        ayah.__init__(self, nama_ayah, nama_kakek)

    def print_nama(self):
        print('nama kakek :', self.nama_kakek)
        print("nama ayah :", self.nama_ayah)
        print("nama anak :", self.nama_anak)

s1 = anak('Juhny', 'Juhna', 'Johny')
print(s1.nama_kakek)
s1.print_nama()

```

Gambar 3.36 Source Code Multilevel Inheritance

OUTPUT

```

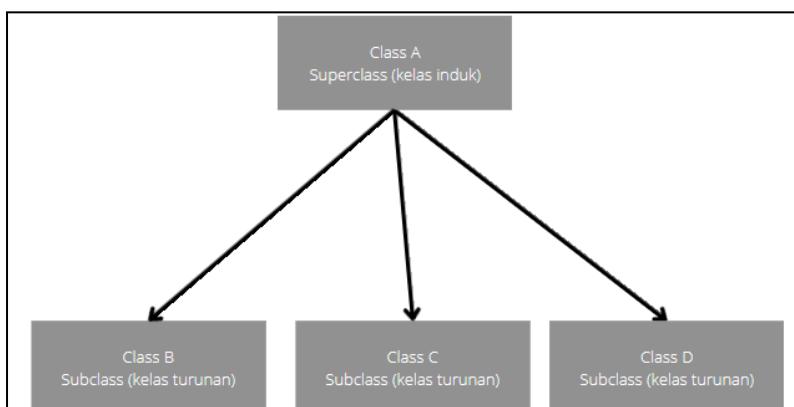
Johny
nama kakek : Johny
nama ayah : Juhna
nama anak : Juhny

```

Gambar 3.37 Output Multilevel Inheritance

d. Hierarchical Inheritance (Pewarisan Hirarki)

Ketika lebih dari satu kelas turunan dibuat dari satu kelas utama, jenis dari pewarisan ini dapat disebut Hierarchical inheritance.



Gambar 3.38 Konsep Hierarchical inheritance

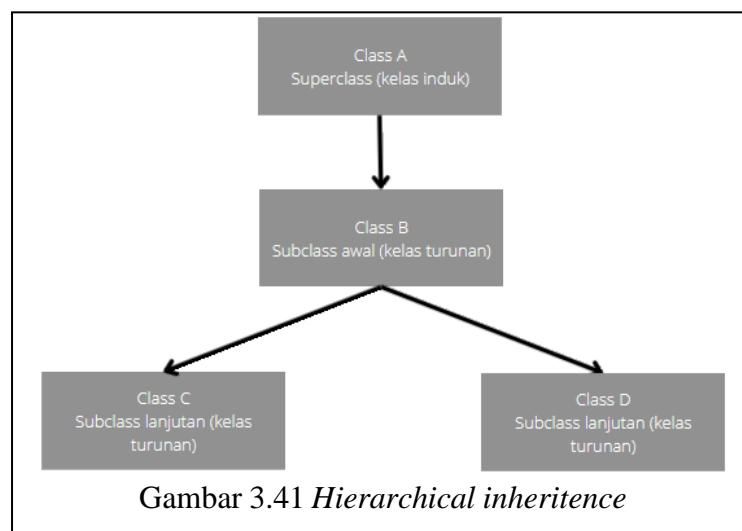
Berikut adalah contoh pengimplementasian *Hierarchical Inheritance*:

Tabel 3.12 Contoh *Hierarchical Inheritance*

SOURCECODE
<pre> ● ● ● class utama: def func1(self): print("fungsi ini ada pada kelas utama") class anak1(utama): def func2(self): print("fungsi ini ada pada kelas anak pertama") class anak2(utama): def func3(self): print("fungsi ini ada pada kelas anak kedua") object1 = anak1() object2 = anak2() object1.func1() object1.func2() object2.func1() object2.func3() </pre>
<p>Gambar 3.39 Sourcecode <i>Hierarchical Inheritance</i></p>
OUTPUT
<pre> fungsi ini ada pada kelas utama fungsi ini ada pada kelas anak pertama fungsi ini ada pada kelas utama fungsi ini ada pada kelas anak kedua </pre>
<p>Gambar 3.40 Output <i>Hierarchical Inheritance</i></p>

e. Hybrid Inheritance (Pewarisan Hibrida)

Jenis ini adalah ketika pewarisan terdiri dari beberapa jenis pewarisan lainnya.



Berikut adalah contoh pengimplementasian *Hybrid Inheritance*:

Tabel 3.13 Contoh *Hybrid Inheritance*

SOURCECODE

```
class utama:
    def func1(self):
        print("fungsi ini ada pada kelas utama")

class anak1(utama):
    def func2(self):
        print("fungsi ini ada pada kelas anak pertama")

class anak2(utama):
    def func3(self):
        print("fungsi ini ada pada kelas anak kedua")

class anak3(anak1, utama):
    def func4(self):
        print("fungsi ini ada pada kelas anak ketiga")

object = anak3()
object.func1()
object.func2()|
```

Gambar 3.42 Source Code Hybrid Inheritance

OUTPUT

```
fungsi ini ada pada kelas utama
fungsi ini ada pada kelas anak pertama
```

Gambar 3.43 Output Hybrid Inheritance

Modul 4: Polymorphism & Encapsulation

4.1 Tujuan Praktikum

1. Memahami konsep *polymorphism* dan *encapsulation* pada program dengan menggunakan Bahasa Python.
2. Mengimplementasikan konsep *polymorphism* dan *encapsulation* pada program.
3. Memahami perbedaan *polymorphism* dan *encapsulation*.

4.2 Alat dan Bahan

1. PC yang terinstall Python 3
2. PC yang terinstall IDE PyCharm / Visual Studio Code

4.3 Dasar Teori

4.3.1 Polymorphism

Polymorphism berasal dari bahasa Yunani yang terdiri dari 2 kata, yaitu *Poly* (banyak) dan *Morphis* (satu bentuk). Dilihat dari asal katanya, *Polymorphism* dapat diartikan sebagai kemampuan dari suatu method yang di mana didalam *method* ini terdapat banyak aksi yang berbeda-beda. Dalam PBO, konsep ini memungkinkan user untuk melakukan suatu aksi yang sama, tetapi proses yang berbeda.

Keuntungan menggunakan program dengan konsep *Polymorphism* adalah dapat menggunakan *class - class* yang kita buat sebagai *superclass* (induk *class*), dan membuat *class* baru berdasarkan *superclass* tersebut dengan karakteristik yang lebih khusus dari *behavior* (tingkah laku) umum yang dimiliki *superclass*, sehingga tidak perlu menuliskan kode dari nol atau pun mengulangnya, tetapi tetap bisa menambahkan atribut dan atau metode unik dari *class* itu sendiri.

Konsep *Polymorphism* dapat diterapkan pada dua metode yaitu *overloading* dan *overriding*.

A. Metode Overriding

Metode *Overriding* adalah penggunaan metode dari *class* yang lain dengan nama metode yang sama dengan induk *class* tetapi memiliki isi program atau argumen yang berbeda dengan kelas induknya. Berikut contoh program dari metode *Overriding*:

Tabel 4.1 *Polymorphism – Overriding*

SOURCE CODE

The screenshot shows a Python code editor with a dark theme. At the top, there are three colored dots (red, yellow, green). Below them is the Python code:

```
class Parent(object):
    def override(self):
        print("Ini adalah fungsi override() dari kelas Parent")

class Child(Parent):
    def override(self):
        print("Ini adalah fungsi override() dari kelas Child")

class Child2(Parent):
    def override(self):
        print("ini adalah fungsi override() dari kelas Child2")

dad = Parent()
son = Child()
daughter = Child2()

dad.override()
son.override()
daughter.override()
```

Gambar 4.1 *Source Code Polymorphism – Overriding*

OUTPUT

The screenshot shows a terminal window with a dark theme. At the top, there are tabs: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is underlined), and JUPYTER. The terminal output is as follows:

```
PS C:\vscode\python> python Polymorphism.py
Ini adalah fungsi override() dari kelas Parent
Ini adalah fungsi override() dari kelas Child
Ini adalah fungsi override() dari kelas Child2
PS C:\vscode\python>
```

Gambar 4.2 *Output Program Polymorphism – Overriding*

B. Metode *Overloading*

Overloading adalah penggunaan metode di suatu *class* yang memiliki metode yang sama tetapi memiliki isi program atau argumen yang berbeda. Namun sayangnya Python sebagai bahasa pemrograman yang dinamik secara *default* Python tidak mendukung penggunaan metode *overloading*. Masalah dari metode *overloading* pada Python adalah kita bisa saja melakukan *overload* metode tapi kita hanya dapat menggunakan metode yang paling akhir didefinisikan. Mari kita lihat penggunaan metode *overloading* pada Python di bawah ini:

Tabel 4.2 *Polymorphism - Overloading Error*

SOURCE CODE
<pre>● ● ● class Penjumlahan(): def add(self, a, b): return a + b def add(self, a, b, c): return a + b + c obj_1 = Penjumlahan() print(obj_1.add(3,1,3)) print(obj_1.add(3,1))</pre>

Gambar 4.3 Source Code Polymorphism - Overloading Error

OUTPUT
<pre>PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER PS C:\vscode\python> python Polymorphism.py 7 Traceback (most recent call last): File "C:\vscode\python\Polymorphism.py", line 35, in <module> print(obj_1.add(3,1)) TypeError: Penjumlahan.add() missing 1 required positional argument: 'c' PS C:\vscode\python> </pre>

Gambar 4.4 Output Program Polymorphism - Overloading Error

Pada contoh kode diatas kita telah mendefinisikan dua metode dengan nama yang sama tetapi menerima parameter yang berbeda. Hasilnya apabila dijalankan, Python akan mengembalikan *error* seperti yang terlihat pada tabel *output* diatas yang menunjukkan *error* pada *line* 35, menunjukkan bahwasannya kita hanya dapat menggunakan metode yang kita definisikan terakhir yaitu metode *add* yang menerima tiga parameter, pemanggilan metode yang lain akan menghasilkan *error*. Untuk mengatasi masalah tersebut kita dapat menggunakan solusi contohnya seperti berikut:

Tabel 4.3 *Polymorphism - Overloading Solution*

SOURCE CODE

```
class Penjumlahan():
    def add(self, *args):
        result = 0
        for i in args:
            # cek apakah argumen bertipe int
            if type(i == int):
                result += 1
        return result

obj_1 = Penjumlahan()

print(obj_1.add(3, 1, 3))
print(obj_1.add(3, 1))
```

Gambar 4.5 Code Polymorphism - Overloading Solution

OUTPUT

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

PS C:\vscode\python> python Polymorphism.py
7
4
PS C:\vscode\python>
```

Gambar 4.6 Output Program Polymorphism - Overloading Solution

4.3.2 Kelas Abstrak

Kelas Abstrak merupakan kelas yang didalamnya terdapat satu atau lebih metode abstrak. Sebuah metode abstrak merupakan metode yang dideklarasikan tapi tidak memiliki implementasi apapun. Lebih lanjut lagi sebuah kelas abstrak bisa dibilang merupakan sebuah *blueprint* untuk membuat kelas yang lain. Sehingga *instance* dari sebuah kelas abstrak bukan merupakan objek melainkan sebuah kelas lainnya. Python secara *default* sebenarnya tidak menyediakan kelas abstrak, namun terdapat modul yang dapat digunakan untuk membuat kelas abstrak yaitu modul ABC atau *abstract base class*. Berikut adalah implementasi kelas abstrak pada Python:

Tabel 4.4 Kelas Abstrak

SOURCE CODE
<pre>● ● ● class Animal(ABC): @abstractmethod # abstract method def jumlahkaki(self): pass class Kucing(Animal): def jumlahkaki(self): # overriding pada metode abstrak print("jumlah kaki kucing ada empat") class Ayam(Animal): def jumlahkaki(self): # overriding pada metode abstrak print("jumlah kaki ayam ada dua") Minji = Kucing() Mino = Ayam() Minji.jumlahkaki() Mino.jumlahkaki()</pre>

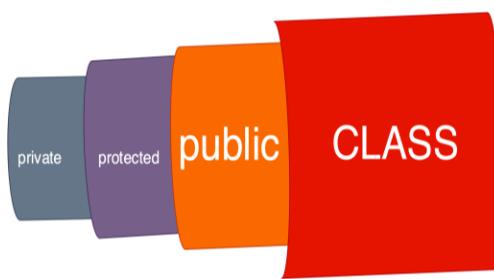
Gambar 4.7 Source Code Polymorphism - Overloading Solution

OUTPUT
<pre>PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER PS C:\vscode\python> python KelasAbstrak.py jumlah kaki kucing ada empat Jumlah Kaki ayam ada dua PS C:\vscode\python> █</pre>

Gambar 4.8 Output Program Polymorphism - Overloading Solution

4.3.3 *Encapsulation* (Enkapsulasi)

Konsep pembungkusan dalam OOP adalah sebuah proses atau cara menyembunyikan informasi dari suatu kelas program sehingga tidak dapat diinterfensi oleh program lain. Dengan demikian, kita dapat membuat program yang terintegrasi, tanpa harus mendeklarasikan variabel-variabel yang bersifat eksternal.



Gambar 4.9 Enkapsulasi

Berikut ini adalah contoh implementasi dari enkapsulasi pada Python:

Tabel 4.5 Implementasi Enkapsulasi

SOURCE CODE
<pre> ● ● ● class Enkapsulasi(): def __init__(self,nama,Balance,noRekening): self.nama = nama self.__Saldoku = Balance self.NoRekening = noRekening @property def Balance(self): pass @Balance.setter def Balance(self,input): self.__Saldoku = input @Balance.getter def Balance(self): return self.__Saldoku Rama = Enkapsulasi('Rama',1000,1009900) print('Menampilkan nama pengguna : %s'%Rama.nama) print('Menampilkan no rekening pengguna : %s '%Rama.NoRekening) print('\nCara mengubah saldo melalui metode set, mengganti saldo menjadi 10000000') Rama.Balance = 10000000 print('cara mengetahui saldo pengguna') print('\nSaldo rekening pengguna atasnama {} adalah {}'.format(Rama.nama,Rama.Balance)) </pre>

Gambar 4.10 Source Code Implementasi Enkapsulasi

OUTPUT
<pre> PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER PS C:\vscode\python> python Enkapsulasi.py Menampilkan nama pengguna : Rama Menampilkan no rekening pengguna : 1009900 Cara mengubah saldo melalui metode set, mengganti saldo menjadi 10000000 Cara mengetahui saldo pengguna Saldo rekening pengguna atasnama Rama adalah 10000000 PS C:\vscode\python> </pre>

Gambar 4.11 Output Program Implementasi Enkapsulasi

Modul 5: Koneksi Database MySQL

5.1 Tujuan Praktikum

1. Memahami apa itu MySQL dan kegunaannya.
2. Mengimplementasikan konsep database MySQL pada program.
3. Memahami cara membuat dan menggunakan database MySQL.

5.2 Alat dan Bahan

1. PC yang terinstall Python 3
2. PC yang terinstall IDE PyCharm / Visual Studio Code

5.3 Dasar Teori

5.3.1 Database MySQL

MySQL sendiri adalah sebuah *database management system* yang menggunakan perintah dasar SQL (*Structured Query Language*) yang bersifat *open source*. MySQL juga merupakan salah satu system management database yang biasa digunakan untuk mengelola berbagai macam informasi dari sebuah website yang membutuhkan database server atau hosting. MySQL memiliki antarmuka atau *interface* yang Bernama PHPMyAdmin.

Keunggulan dari MySQL itu sendiri tidak membutuhkan RAM yang besar, sangat kompatibel dengan bahasa pemrograman lain dan yang struktur data yang fleksibel. Serta kekurangan dari MySQL adalah tidak cocok digunakan untuk database yang besar.

5.3.2 Menghubungkan Python dengan MySQL

A. Instalasi mysql-connector

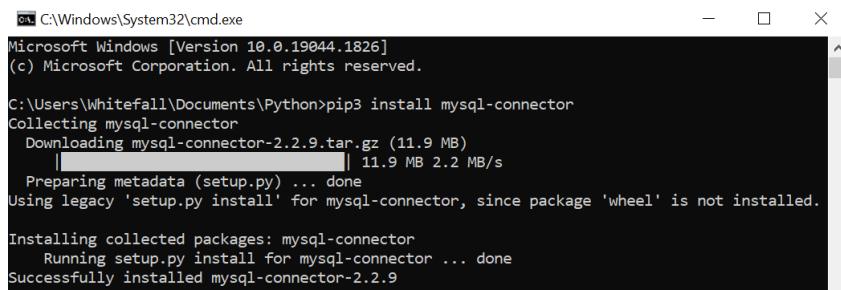
Untuk menghubungkan aplikasi python yang kita buat ke *database* MySQL, pertama kita harus menginstal sebuah modul yang bernama “mysql-connector”. Cara penginstalannya adalah dengan mengetikan kode di bawah ke *cmd* atau *terminal* kalian.



```
pip3 install mysql-connector
```

Gambar 5.1 Perintah *install* mysql-connector

Nanti akan muncul tampilan seperti dibawah, kemudian tunggu sampai muncul tulisan “Successfully installed”. Apabila modul mysql-connector sudah berhasil di instal maka kita sudah bisa membuat koneksi ke database.



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19044.1826]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Whitefall\Documents\Python>pip3 install mysql-connector
Collecting mysql-connector
  Downloading mysql-connector-2.2.9.tar.gz (11.9 MB)
    |████████| 11.9 MB 2.2 MB/s
  Preparing metadata (setup.py) ... done
Using legacy 'setup.py install' for mysql-connector, since package 'wheel' is not installed.

Installing collected packages: mysql-connector
  Running setup.py install for mysql-connector ... done
Successfully installed mysql-connector-2.2.9
```

Gambar 5.2 Proses *install mysql-connector*

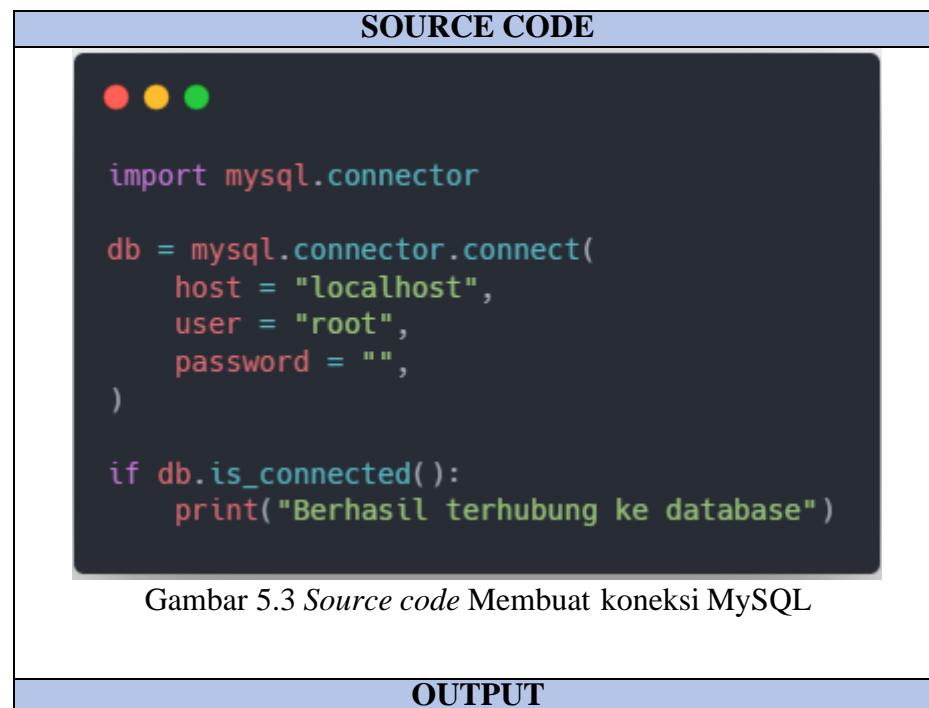
B. Membuat koneksi ke MySQL

Setelah selesai menginstal modul mysql-connector, untuk membuat koneksi ke MySQL kita harus mengimport modul tersebut kemudian membuat sebuah objek database dengan cara memanggil fungsi “`mysql.connector.connect()`” dan mengisi parameternya dengan kredensial antara lain adalah *host*, *user*, dan *password* database kita.

Secara *default*, nilai *host* adalah “**localhost**”, *username* atau *user* adalah “**root**” dan *password*-nya **kosong**. Untuk lebih jelasnya mengenai inisialisasi objek database, perhatikan *source code* dibawah ini:

Tabel 5.1 Koneksi MySQL

SOURCE CODE
<pre>● ● ● import mysql.connector db = mysql.connector.connect(host = "localhost", user = "root", password = "",) if db.is_connected(): print("Berhasil terhubung ke database")</pre>
OUTPUT



```
Berhasil terhubung ke database
```

Gambar 5.3 *Source code* Membuat koneksi MySQL

```
C:\Users\62812\OneDrive\Documents\SOURCE CODE\SEA>python -u  
"c:\Users\62812\OneDrive\Documents\SOURCE CODE\SEA\test.py"  
Berhasil terhubung ke database
```

Gambar 5.4 *Output* Membuat koneksi MySQL

Untuk mengecek apabila kita berhasil terhubung ke MySQL, kita dapat memanggil fungsi “`is_connected()`” pada objek database dan memasukkannya ke dalam *if-statement*. Apabila kredensial yang kalian masukan pada fungsi “`mysql.connector.connect()`” sudah benar maka fungsi “`is_connected()`” akan mengembalikan nilai *true* sedangkan apabila kredensial kalian tidak valid maka nilai yang dikembalikan adalah *false*.

C. Membuat database baru

Setelah kita terkoneksi ke MySQL Langkah selanjutnya adalah inisialisasi objek *database*, kita perlu menambahkan nama *database* yang akan kita gunakan pada kredensial. Dengan *query* “`CREATE DATABASE nama_database`”

Tabel 5.2 Membuat Database

SOURCE CODE
<pre>import mysql.connector db = mysql.connector.connect(host="localhost", user="root", password="") cursor = db.cursor() cursor.execute("CREATE DATABASE seantuy") print("Database berhasil dibuat!")</pre>

Gambar 5.5 *code* Membuat Database

OUTPUT
<pre>C:\Users\62812\OneDrive\Documents\SOURCE CODE\SEA>python -u "c:\Users\62812\OneDrive\Documents\SOURCE CODE\SEA\create_db .py" Database berhasil dibuat!</pre>

Gambar 5.6 *Output* Membuat Database

Pada contoh di atas nama *database* yang digunakan adalah “seantuy”. Kemudian kita perlu membuat objek “cursor”, objek inilah yang nantinya akan mengeksekusi perintah ke *database*. Setelah itu kita definisikan perintah dan data yang akan di eksekusi seperti contoh di atas. Pendefinisian perintah atau *query* boleh dengan *string* biasa atau dengan *string format* yang isi datanya berupa *tuple* seperti contoh di atas.

D. Mengecek database

Pada Langkah ini kita hanya mengecek ke database localhost yang dapat diakses di <http://localhost/phpmyadmin> ketika sudah berhasil menambahkan database maka akan keluar output berikut, dimana pada database “seantuy” masih tidak memiliki tabel sama sekali

Tabel 5.3 Cek Database

Gambar 5.7 Output Cek Database

5.3.3 CRUD (Create, Read, Update, Delete)

CRUD adalah singkatan yang berasal dari Create, Read, Update, dan Delete, dimana keempat istilah tersebut merupakan perintah utama yang nantinya diimplementasikan ke dalam basis data.

A. Create

Create merupakan sebuah proses membuat tabel baru ke dalam database. Pada MySQL proses create dilakukan dengan menggunakan query “CREATE TABLE nama_tabel (nama_kolom), VALUES (data)”. Untuk lebih jelasnya silakan perhatikan contoh implementasi Create pada python di bawah ini:

Tabel 5.4 *Create - Database*

SOURCE CODE
<pre>● ● ● import mysql.connector db = mysql.connector.connect(host = "localhost", user = "root", password = "", database = "seantuy",) cursor = db.cursor() sql = """CREATE TABLE asisten_lab (aslab_id INT AUTO_INCREMENT PRIMARY KEY, nama VARCHAR(255), kode Varchar(255)) """ cursor.execute(sql) print("Tabel asisten lab berhasil dibuat!")</pre>
OUTPUT

```
C:\Users\62812\OneDrive\Documents\SOURCE CODE\SEA>C:/User  
ppData/Local/Microsoft/WindowsApps/python3.9.exe "c:/User  
neDrive/Documents/SOURCE CODE/SEA/create_table.py"  
Tabel asisten lab berhasil dibuat!
```

Gambar 5.9 Implementasi *Create*

Untuk mengeksekusi data kita gunakan fungsi “execute” yang ada di objek “cursor”. Apabila kita menggunakan *string* biasa maka isi parameter “execute” langsung ditulis “cursor.execute(sql)” tidak perlu ditambah data di sebelahnya. Pada tahap ini *query* hanya dicek apakah ada *error* atau tidak dan data belum masuk ke *database*. Apabila tidak ada *error* pada *query* ketika di eksekusi maka kita bisa lanjut menyimpan perubahan menggunakan fungsi “commit” pada objek database. Ketika *query* sudah di *commit* maka data data yang kita masukan sudah tersimpan pada *database*.

B. Insert

Walaupun CRUD adalah singkatan yang berasal dari Create, Read, Update, dan Delete, namun ada satu fungsi lagi yang tidak kalah penting yaitu insert karena sebelum kita menginsert sebuah data, maka kita tidak mungkin bisa meread database yang kosong.

Insert sendiri merupakan sebuah proses memasukkan data baru kedalam tabel yang sudah dibuat pada proses create sebelumnya. Pada MySQL proses insert dilakukan menggunakan query “INSERT INTO nama_tabel (nama_kolom), VALUES (data)”.

Tabel 5.5 *Insert* - Database

SOURCE CODE

```
import mysql.connector

db = mysql.connector.connect(
    host = "localhost",
    user = "root",
    password = "",
    database = "seantuy",
)

cursor = db.cursor()
sql = "INSERT INTO asisten_lab (nama, kode) VALUES (%s, %s)"
values = [
    ("Rizky", "RIY"),
    ("Malik", "MAL"),
    ("Mario", "RIO"),
    ("Hitler", "HIT")
]
for val in values:
    cursor.execute(sql, val)
db.commit()

print("{} data ditambahkan".format(len(values)))
```

Gambar 5.10 *Source code* Implementasi Insert

Pada contoh kali ini adalah insert yang menggunakan perulangan karena kita ingin memasukkan data yang lebih dari satu, tidak lupa kita menggunakan *commit* karena disini kita merubah apa isi tabel itu sendiri.

Tabel 5.6 *Output* Implementasi Insert

OUTPUT
C:\Users\62812\OneDrive\Documents\SOURCE CODE\SEA>C:/User ppData/Local/Microsoft/WindowsApps/python3.9.exe "c:/User neDrive/Documents/SOURCE CODE/SEA/insert_one.py" 4 data ditambahkan

Gambar 5.11 *Output* Insert

			aslab_id	nama	kode
<input type="checkbox"/>	 Edit	 Copy	 Delete	1	Rizky RIY
<input type="checkbox"/>	 Edit	 Copy	 Delete	2	Malik MAL
<input type="checkbox"/>	 Edit	 Copy	 Delete	3	Mario RIO
<input type="checkbox"/>	 Edit	 Copy	 Delete	4	Hitler HIT

Gambar 5.12 Output Implementasi Insert

Pada bagian output ini kita bisa melihat di database MySQL bahwa data yang kita insert berhasil ditambahkan kedalam tabel.

C. Read

Fungsi yang ketiga yaitu read, memungkinkan kita untuk mencari atau mengambil data tertentu yang berada di dalam tabel. Query yang digunakan untuk menjalankan fungsi ini adalah “SELECT nama_kolom FROM nama_tabel”. Pada fungsi ini juga kita dapat mengaplikasikan “search” dengan menggunakan klausa “WHERE” pada akhir *query* “SELECT”. Untuk pengaplikasiannya silakan perhatikan *source code* di bawah ini:

Tabel 5.7 Read - Database

SOURCE CODE

```
import mysql.connector

db = mysql.connector.connect(
    host = "localhost",
    user = "root",
    password = "",
    database = "seantuy",
)

cursor = db.cursor()
sql = "SELECT * FROM asisten_lab"
cursor.execute(sql)

results = cursor.fetchall()

for data in results:
    print(data)
```

Gambar 5.13 *Source code* Implementasi *Read*

OUTPUT

```
[1]: execute(query)
(1, 'Rizky', 'RIY')
(2, 'Malik', 'MAL')
(3, 'Mario', 'RIO')
(4, 'Hitler', 'HIT')
```

Gambar 5.14 *Output* Implementasi *Read*

Pada *source code* di atas setelah melakukan eksekusi kita bisa langsung mengambil datanya menggunakan fungsi “fetchall” pada objek cursor. Data yang diambil berupa tuple dan disimpan pada variabel result. Pada proses ini kita tidak perlu melakukan commit karena kita tidak melakukan perubahan apa pun pada tabel.

Untuk proses read ada fungsi lain yang dapat digunakan yaitu “fetchmany(jumlah yang ingin dibaca)” dan “fetchone”. Dimana fetchmany digunakan untuk membaca jumlah tertentu, dan fetchone digunakan untuk membaca satu baris saja.

D. Update

Update sendiri merupakan proses dimana kita merubah sebuah data pada tabel, kolom, atau baris tertentu di dalam database. Pada MySQL proses update dilakukan menggunakan query “UPDATE nama_tabel SET nama_kolom=(jenis data) WHERE primary_id=(jenis data).

Disini kita tidak terbatas hanya mengupdate satu data saja, tetapi kita dapat mengupdate banyak data menggunakan cara yang tidak beda jauh dengan cara insert di atas. Tidak lupa untuk *commit* karena kita merubah isi tabel di dalam database.

Tabel 5.8 *Update* - Database

SOURCE CODE
<pre>import mysql.connector db = mysql.connector.connect(host = "localhost", user = "root", password = "", database = "seantuy",) cursor = db.cursor() sql = "UPDATE asisten_lab SET nama=%s, kode=%s WHERE aslab_id=%s" val = ("Hirohito", "ITO", 1) cursor.execute(sql, val) db.commit() print("{} data diubah".format(cursor.rowcount))</pre>
Gambar 5.15 <i>Source code</i> Implementasi <i>Update</i>
OUTPUT
<pre>DE\SEA\update.py" " 1 data diubah</pre>
Gambar 5.16 <i>Output Update</i>

	<input type="button" value="←"/>	<input type="button" value="→"/>		aslab_id	nama	kode
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	1	Hirohito	ITO
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	2	Malik	MAL
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	3	Mario	RIO
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	4	Hitler	HIT

Gambar 5.17 Output Implementasi Update

Kita dapat melihat bagaimana hasil dari perubahan yang kita buat pada database secara langsung seperti pada hasil output di atas.

E. Delete

Delete adalah fungsi terakhir yang kita pelajari pada modul kali ini, dimana delete itu sendiri merupakan proses kita menghapus sebuah data pada sebuah tabel, kolom, atau baris tertentu di dalam database. Pada MySQL proses update dilakukan menggunakan *query* “DELETE FROM nama_tabel WHERE primary_id”.

Tabel 5.9 Delete - Database

SOURCE CODE

```
import mysql.connector

db = mysql.connector.connect(
    host = "localhost",
    user = "root",
    password = "",
    database = "seantuy",
)

cursor = db.cursor()
sql = "DELETE FROM asisten_lab WHERE aslab_id=%s"
val = (1, )
cursor.execute(sql, val)

db.commit()

print("{} data dihapus".format(cursor.rowcount))
```

Gambar 5.18 *Source code* Implementasi *Delete*

OUTPUT

```
E CODE\SEA\delete.py" "
1 data dihapus
```

Gambar 5.19 *Output Delete*

	← ↑ →		▼	aslab_id	nama	kode
<input type="checkbox"/>	 Edit	 Copy	 Delete	2	Malik	MAL
<input type="checkbox"/>	 Edit	 Copy	 Delete	3	Mario	RIO
<input type="checkbox"/>	 Edit	 Copy	 Delete	4	Hitler	HIT

Gambar 5.20 *Output Implementasi Delete*

Modul 6: Web Scraping

6.1 Tujuan Praktikum

1. Memahami konsep *web scraping* sederhana menggunakan python.
2. Mengimplementasikan konsep *web scraping* pada program.
3. Mampu mengolah data hasil *scraping* secara sederhana.

6.2 Alat dan Bahan

1. PC yang terinstall Python 3
2. PC yang terinstall IDE PyCharm / Visual Studio Code
3. PC yang terinstall *library BeautifulSoup* dan *requests*

6.3 Dasar Teori

6.3.1 Konsep Web Scraping

Web Scraping adalah proses pengumpulan data *web* terstruktur yang dilakukan secara otomatis dengan menggunakan aplikasi atau kode pemrograman khusus. Ada beberapa teknik *Web Scrapping* yang biasa dilakukan; menggunakan *regular expression*, *parsing* HTML, menggunakan XPATH dan lainnya, namun untuk praktikum kali ini yang akan dibahas hanya teknik *Parsing* HTML.

Parsing HTML sendiri adalah teknik Scaping dengan cara mengirimkan HTTP *request* kepada server penyimpan data pada website yang ingin diambil datanya dan menelusuri setiap elemen HTML sampai menemukan data yang diinginkan. Metode ini dapat dilakukan pada data website statis maupun dinamis. Metode ini juga memungkinkan kita mengambil data dalam jumlah banyak. Namun seringkali website menghalang laman mereka agar sulit untuk di ambil datanya ataupun memblokir jika terlalu sering melakukan *request*.

6.3.2 Library Requests dan Beautiful Soup

Dalam *Web Scrapping*, kita bisa menggunakan bahasa Python. Dalam Python, *web scraping* dapat dilakukan dengan menggunakan *library requests* dan *Beautiful Soup*. *Library Request* digunakan untuk mengirimkan HTTP *request*. Sedangkan, *BeautifulSoup* digunakan untuk mengambil elemen HTML *web*.

Requests adalah library Python yang dapat digunakan untuk mengirimkan HTTP *request*. *Library* ini digunakan untuk memudahkan dalam mengambil data HTML dari *web*. *Requests* mengembalikan data berupa objek respon dengan seluruh data respon.

BeautifulSoup merupakan library Python yang dapat digunakan untuk *Parsing* HTML dan XML. *BeautifulSoup* bekerja dengan *parser* bawaan dari

python (`html.parser`), atau parser lain seperti `parser LXML` dan `HTML5lib` untuk mempermudah dalam web scrapping melalui bahasa python.

B. Fungsi `requests.get`

Dalam melakukan *web scraping* kita perlu melakukan HTTP *request*. Dalam library `requests` fungsi `requests.get` yang digunakan untuk melakukan HTTP *request*. Kita dapat mengirimkan HTTP request menggunakan syntax berikut:

Tabel 6.1 Fungsi *Request.get*

SOURCE CODE
<pre>#import Requests import requests #request ke laman web resp = requests.get('https://id.wikipedia.org/wiki/Michael_Bambang_Hartono') #print respon print(resp)</pre>
Gambar 6.1 Source Code <i>Requests.get</i>

OUTPUT
<Response [200]> : □
Gambar 6.2 Output Program <i>Requests.get</i>

Fungsi `requests.get` akan menghasilkan output “<Response [200]>” bila website mengijinkan kita melakukan *scraping*. Namun, beberapa website bila dilakukan *request* akan menghasilkan angka yang berbeda yang dapat berarti HTTP *request* kita ditolak.

C. Inisiasi `Beautifulsoup`

a. `.content`

Fungsi `.content` digunakan untuk mengambil isi (konten) dari fungsi `requests.get` yang telah berhasil didapatkan.

b. *html.parser*

Fungsi `html.parser` digunakan untuk menguraikan (*parsing*) HTML dari *web* yang telah berhasil *request*.

Perhatikan contoh berikut:

Tabel 6.2 Inisiasi *Beautifulsoup*

SOURCE CODE
<pre>● ● ● soup = BeautifulSoup(resp.content, 'html.parser') print(soup)</pre>
Gambar 6.3 <i>Code Inisiasi BeautifulSoup</i>
OUTPUT
<pre><!DOCTYPE html> <html class="client-nojs" dir="ltr" lang="id"> <head> <meta charset="utf-8"/> <title>Michael Bambang Hartono - Wikipedia bahasa Indonesia, ensiklopedia bebas</title></pre>
Gambar 6.4 <i>Program Inisiasi BeautifulSoup</i>

D. Metode dalam *BeautifulSoup*

Perhatikan contoh HTML berikut:

Tabel 6.3 Metode *BeautifulSoup*

SOURCE CODE
<pre>● ● ● <html> <head> <title> The Dormouse's story </title> </head></pre>

```

<body>
  <p class="title">
    <b>
      The Dormouse's story
    </b>
  </p>
  <p class="story">
    Once upon a time there were three little sisters; and their names were
    <a class="sister" href="http://example.com/elsie" id="link1">
      Elsie
    </a>
    ,
    <a class="sister" href="http://example.com/lacie" id="link2">
      Lacie
    </a>
    and
    <a class="sister" href="http://example.com/tillie" id="link3">
      Tillie
    </a>
    ; and they lived at the bottom of a well.
  </p>
  <p class="story">
    ...
  </p>
</body>
</html>

```

Gambar 6.5 Contoh HTML 1

a. *find(tag, attrs)*

Metode *find()* menelusuri dokumen dan akan mengembalikan hasil pertama yang ditemukan. Metode ini dapat menerima 2 argumen utama yaitu nama tag dan atribut dari tag tersebut. Tag merupakan nama tag yang ingin dicari dan atribut adalah sebuah dictionary untuk pencarian yang lebih spesifik. Penggunaan metode *find* dapat dilihat pada contoh berikut:

Tabel 6.4 Fungsi *find()* *BeautifulSoup*

SOURCE CODE
<pre> ● ● ● elemen_a = soup.find('a', {'class': 'sister'}) print(elemen_a) </pre>
<p>Gambar 6.6 Source Code Penggunaan Metode <i>find()</i></p>
OUTPUT
<pre> Elsie > ■ </pre>
<p>Gambar 6.7 Program Penggunaan Metode <i>find()</i></p>

b. *find_all(tag, attrs)*

Metode `find_all()` mirip dengan metode `find()`, namun mengambil semua hasil yang sesuai dan memasukannya kedalam list. Penggunaan metode `find_all` dapat dilihat pada contoh berikut:

Tabel 6.5 Fungsi `find_all()` *BeautifulSoup*

SOURCE CODE
<pre>● ● ● semua_elemen_a = soup.find_all('a', {'class': 'sister'}) print(semua_elemen_a)</pre>
Gambar 6.8 Source Code Penggunaan Metode <code>find_all()</code>
OUTPUT
<pre>[Elsie , Lacie , Tillie]> : </pre>
Gambar 6.9 Output Program Penggunaan Metode <code>find_all()</code>

c. *get_text()*

Metode `get_text()` digunakan untuk mengambil semua teks yang berada di dalam dokumen atau tag. Contoh penggunaan metode `get_text` dapat dilihat pada *source code* berikut:

Tabel 6.6 Fungsi `get_text()` *BeautifulSoup*

SOURCE CODE
<pre>● ● ● semua_elemen_a = soup.find_all('a', {'class': 'sister'}) for elemen_a in semua_elemen_a: print(elemen_a.get_text())</pre>
Gambar 6.10 Source Code Penggunaan Metode <code>get_text()</code>
OUTPUT
<pre>Elsie Lacie Tillie :> </pre>
Gambar 6.11 Output Program Penggunaan Metode <code>get_text()</code>

E. Penerapan library *requests* dan *BeautifulSoup*

Dalam penerapannya secara sederhana kita dapat melakukan HTTP *request* dengan fungsi *request.get* terlebih dahulu. Setelah itu, jika setelah kita tes dengan *print request* mendapatkan respon 200 maka kita dapat lanjutkan dengan html parsing menggunakan *BeautifulSoup* (inisiasi *BeautifulSoup*). Jika sudah berhasil kita dapat mencari data yang kita inginkan dengan fungsi-fungsi dari *BeautifulSoup* seperti *find*, *find_all*, dan *get_text*. Contoh sederhana penggunaan *requests* dan *BeautifulSoup* dalam *web scraping*:

Tabel 6.7 *Web Scraping* Sederhana

SOURCE CODE
<pre>#import BeautifulSoup dan Requests from bs4 import BeautifulSoup import requests #request ke laman web resp = requests.get('https://id.wikipedia.org/wiki/Michael_Bambang_Hartono') #inisialisasi BeautifulSoup soup = BeautifulSoup(resp.content, 'html.parser') #mencari header h1 dengan id firstheading (header/judul pertama) print(soup.find('h1', {'id': 'firstHeading'}).get_text())</pre>

OUTPUT
<pre>Michael Bambang Hartono</pre>

Gambar 6.12 *Source Code Web Scraping* Sederhana

Gambar 6.13 *Output Program Web Scraping* Sederhana

Dalam contoh program di atas, program mengambil data *header* dari halaman biografi Michael Bambang Hartono di wikipedia. Data diambil dengan mengirimkan HTTP *request* dengan *requests* lalu melakukan *parsing* dengan *BeautifulSoup*. Pada hasil *parsing* HTML dilakukan pencarian header yang berisi data nama.

6.3.3 Mengolah Data

Dalam *web scraping*, kita sering kali menemui data berupa tabel. Data tabel biasanya perlu dilakukan pengolahan terlebih dahulu sebelum bisa digunakan pada

program *scraping*. Data tabel yang sudah diolah akan menghasilkan output seperti *dictionary* atau *list*. Pengolahan data tabel web scraping dapat dilihat pada contoh di bawah ini:

Tabel 6.8 *Web Scraping* pada Tabel

SOURCE CODE
<pre>● ● ● from bs4 import BeautifulSoup import requests # request ke laman web resp = requests.get('https://id.wikipedia.org/wiki/Robert_Budi_Hartono') # inisialisasi BeautifulSoup soup = BeautifulSoup(resp.content, 'html.parser') # cari tabel dengan class 'infobox biography vcard' infobox = soup.find('table', {'class': 'infobox biography vcard'}) # ambil semua baris data = infobox.find_all('tr') informasi = {} for data in data[2:]: # untuk setiap data kecuali 2 data pertama (judul dan gambar) # ambil teks judul data judul = data.find('th').get_text().replace('\xa0', ' ') # ambil teks informasi data info = data.find('td').get_text().replace('\xa0', ' ') # masukan ke dictionary dengan key : judul dan value : info informasi[judul] = info print(informasi)</pre>

Gambar 6.14 *Source Code Web Scraping Tabel*

OUTPUT
<pre>{'Lahir': '28 April 1941 (umur 81)Semarang, Jawa Tengah, Hindia Belanda', 'Nama lain': 'Oei Hwie Tjhong', 'Warga negara': 'Indonesia', 'Pekerjaan': 'Pengusaha', 'Dikenal atas': 'DjarumBank BCAPolytron', 'Kekayaan bersih': 'US\$ 23,3miliar (US\$ 58juta) (11 Maret 2022)[1]', 'Suami/istri': 'Widowati Hartono', 'Anak': 'Victor Rachmat Hartono Martin Basuki HartonoArmand Wahyudi Hartono', 'Orang tua': 'Oei Wie Gwan (ayah)', 'Kerabat': 'Michael Bambang Hartono (kakak)', 'Situs web': 'www.djarum.com'}</pre>

Gambar 6.15 *Output Program Web Scraping*

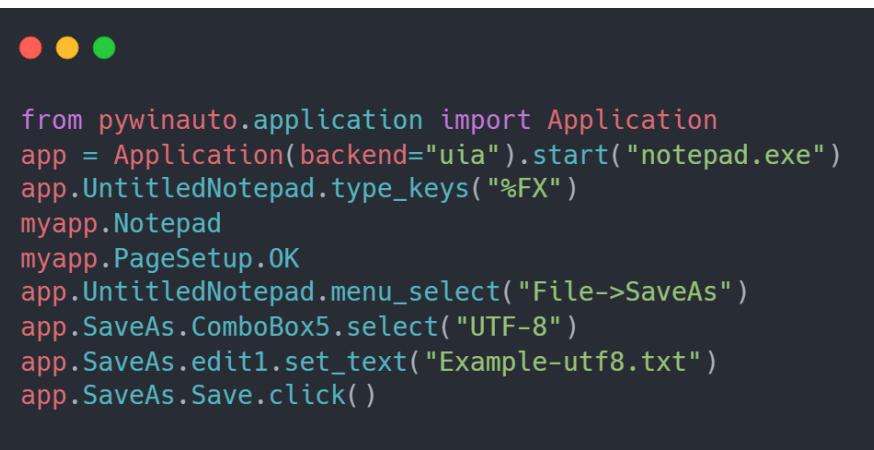
Pada contoh program diatas, program mengambil data tabel informasi singkat tokoh Robert Budi Hartono dari wikipedia. Data yang diambil berbentuk tabel yang berisi biografi singkat yang kemudian dilakukan *scraping* dan diubah kedalam bentuk *dictionary* python.

6.3.4 GUI Automation dengan pywinauto

GUI (*Graphical User Interface*) Automation adalah suatu cara melakukan automasi Input keyboard dan mouse melalui program. Library *pywinauto* adalah salah satu *library* python yang mampu melakukan GUI Automation.

pywinauto menggunakan fungsi *application* yang dapat digunakan untuk berbagai hal. Seperti menggunakan untuk membuka *notepad*. Dalam fungsi itu juga dapat melakukan automasi seperti memasukkan teks dan melakukan *save*.

Tabel 6.9 Pywinauto *notepad*

SOURCE CODE
 <pre>from pywinauto.application import Application app = Application(backend="uia").start("notepad.exe") app.UntitledNotepad.type_keys("%FX") myapp.Notepad myapp.PageSetup.OK app.UntitledNotepad.menu_select("File->SaveAs") app.SaveAs.ComboBox5.select("UTF-8") app.SaveAs.edit1.set_text("Example-utf8.txt") app.SaveAs.Save.click()</pre>

Gambar 6.16 Source Code *pywinauto notepad*

Daftar Pustaka

Asisten Praktikum Lab. RAID, C. S. S., 2021. *MODUL PRAKTIKUM PEMROGRAMAN BERBASIS OBJEK*. Bandung: Laboratorium RAID.

Petanicode, 2019, *Tutorial Python dan MySQL: Membuat Aplikasi CRUDS berbasis teks*. [Online]

Available at: <https://www.petanikode.com/python-mysql/>

Geeksforgeeks, 2021, *Python SQLite – CRUD Operations*. [Online]
Available at: <https://www.geeksforgeeks.org/python-sqlite-crud-operations/>

Geeksforgeeks, 2022, *Implementing Web Scraping in Python with BeautifulSoup*. [Online]

Available at: <https://www.geeksforgeeks.org/implementing-web-scraping-python-beautiful-soup/>

W3School, 2022. *Python Requests Module*. [Online]

Available at: https://www.w3schools.com/python/module_requests.asp

Crummy, 2020, *Beautiful Soup Documentation*. [Online]

Available at: <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

Ngodingdata, 2022. *Tutorial Web Scraping dengan BeautifulSoup di Python*. [Online]

Available at: <https://ngodingdata.com/tutorial-web-scraping-dengan-beautifulsoup-di-python-part-1/#BeautifulSoup>