RESUME PRAKTIKUM
PEMROGRAMAN BERBASIS OBJEK
RB

Disusun oleh :

Fatur Arkan Syawalva

121140229

PROGRAM STUDI TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI SUMATERA
2023

DAFTAR ISI

MODUL 1

1.1 Pengenalan Bahasa Pemrograman Python

Bahasa pemrograman python dibuat oleh Guido Van Rossum pada tahun 1980-an akhir di Centrum Wiskunde & Informatica Belanda. Python mendukung banyak paradigma pemrograman seperti object-oriented, functional dan structured. Bahasa Python menjadi populer sekarang ini dikarenakan sintaks nya yang mudah, didukung oleh library(modul) yang berlimpah dan Python bisa dipakai untuk pemrograman desktop maupun mobile, CLI, GUI, web, otomatisasi, hacking, IoT, robotika, dan lain sebagainya.

Dalam dokumen The Zen of Python (PEP 20), disitu tertera filosofi inti dari bahasa pemrograman Python :

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.

Python sangat mementingkan readability pada kode, untuk mengimplementasikan filosofi itu Python tidak menggunakan kurung kurawal({}) atau keyword (ex. start, begin, end) sebagai gantinya menggunakan spasi(white space) untuk memisahkan blok-blok kode

1.2 Dasar Bahasa Python

1.2.1 Sintaks Dasar

Pada Bahasa python akhir dari sebuah statement adalah baris baru, namun sebuah statement dapat terdiri dari beberapa baris dengan menggunakan backslash (\). Dan dalam pyton tidak menggunakan tanda kurung kurawal seperti pada Bahasa pemrograman C++, namun di grouping blok kode menggunakan spasi atau tab.

1.2.2 Variabel dan Tipe Data Primitif

Tipe Data	Jenis	Nilai
bool	Boolean	True atau false
int	BIlangan Bulat (Integer)	Seluruh bilangan bulat
float	Bilangan real (Float)	Seluruh bilangan real
string	Teks (String)	Kumpulan karakter

1.2.3 Operator

Operator	Nama dan Fungsi	Contoh
+	Penjumlahan	x+y
-	Pengurangan	x-y
*	Perkalian	x*y
/	Pembagian	x/y
**	Pemangkatan	x**y (x^y)
//	Pembagian bulat	x//y
%	Modulus (Sisa pembaian 2 bilangan	x%y
>	Lebih besar dari	x>y
<	Lebih kecil dari	x<y
==	Sama dengan	x==y
!=	Tidak sama dengan	x!=y

>=	Lebih besar atau sama dengan	x>=y
<=	Lebih kecil atau sama dengan	x<=y
=	Menugaskan nilai yang ada di kanan ke operand di sebelah kiri	c= a+b sama a+b=c
+=	Penjumlahan dengan operand yang ditugaskan diletakkan disebelah kanan	c+= a sama c=c+a
-=	Pengurangan dengan operand yang ditugaskan diletakkan disebelah kanan	c-=a sama c=c-a
*=	Perkalian dengan operand yang ditugaskan diletakkan disebelah kanan	c*= a sama c=c*a
/=	Pembagian dengan operand yang ditugaskan diletakkan disebelah kanan	c/= a sama c=c/a
**=	Pemangkatan dengan operand yang ditugaskan diletakkan disebelah kanan	c**= a sama c=c**a
//=	Pembagian bulat dengan operand yang ditugaskan diletakkan disebelah kanan	c//= a sama c=c//a
%=	Operasi sisa bagi dengan operand yang ditugaskan diletakkan disebelah kanan	c%= a sama c=c%a
and	True jika kedua operand benar	x and y
or	True jika salah satu operand benar	x or y
not	True jika operand bernilai salah	not x
&	Bitwise AND	x& y = 0 (0000 0000)
|	Bitwise OR	x | y = 14 (0000 1110)
~	Bitwise NOT	~x = -11 (1111 0101)
^	Bitwise XOR	x ^ y = 14 (0000 1110)
>>	Bitwise right shift	x>> 2 = 2 (0 000 0010)
<<	Bitwise left shift	x<< 2 = 40 (0010 1000)
is	Jika operand yang identik	y is true
is not	Jika operand tidak identik	y is not true
in	Jika nilai/variable terdapat dalam data	1 in x
not in	Jika nilai/variable tidakada dalam data	1 not in x

1.2.4 Tipe data bentukan

Ada 4, dengan perbedaan penggunaan:

- List

  Sebuah kumpulan data yang terurut, dapat diubah, dan memungkinkan ada anggota yang sama
- Tuple

  Sebuah kumpulan data yang terurut, tidak dapat diubah, dan memungkinkan ada anggota yang sama

- Set
  Sebuah kumpulan data yang tidak berurutan, tidak terindeks, dan tidak memungkinkan ada anggota yang sama
- Dictionary
  Sebuah kumpulan data yang tidak berurutan, dapat diubah, tidak memungkinkan ada anggota yang sama

1.2.5 Percabangan

- **IF**

  Contoh:

```
a = int(input())
if (a>0):
    print("positif")
```

```
12
positif
```

- **IF ELSE**

  Contoh:

```
a = int(input())
if (a>0):
    print("positif")
elif (a<0):
    print("negatif")
```

```
-5
negatif
```

- **IF ELSE IF**

  Contoh:

```
a = int(input())
if (a>0):
    print("positif")
elif (a<0):
    print("negatif")
else:
    print("nol")
```

```
0
nol
```

- **NESTED IF**

  Terdapat percabangan dalam percabangan

  Contoh:

```
a = int(input())
if (a>=0):
    if(a==0):
        print("nol")
    else:
        print("bilangan positif")
else:
    print("bilangan negatif")
```

```
-3
bilangan negatif
```

1.2.6 Perulangan

- **Perulangan For**

Sintaks dasar :

```
for i in (kondisi):
```

Contoh perulangan dengan list:

```
Hewan = ["Harimau", "Kelinci", "Kanguru", "Monyet"]
for i in Hewan:
    print(i)
```

```
Harimau
Kelinci
Kanguru
Monyet
```

Contoh perulangan dengan string:

```
for a in "makan":
    print(a)
```

```
m
a
k
a
n
```

Contoh perulangan dengan tuple:

```
bunga = ("sepatu", "matahari", "melati"
for i in bunga:
    print(i)
```

```
sepatu
matahari
melati
```

Sintaks penggunaan range pada for:

1. Satu parameter
   Perulangan akan dilakukan dari indeks 0 sampai kurang dari x.
   ```
   for i in range (x):
   ```
2. Dua parameter
   Perulangan akan dilakukan dari indeks ke-x sampai kurang dari y.
   ```
   for i in range (x,y):
   ```
3. Tiga parameter
   Perulangan akan dilakukan dari indeks ke-x sampai kurang dari y dengan indeks bertambah/increment sejumlah z.
   ```
   for i in range (x,y,z):
   ```

- **Perulangan While**

Sintaks dasar:

```
while(kondisi):
```

Contoh perulangan while:

```
count=int(0)
kalimat=(input())
while(kalimat != '!'):
    if kalimat == 'a':
        count += 1
    kalimat = (input())
print("Jumlah karakter a: ", count)
```

1.2.7 Fungsi

Dengan menggunakkan fungsi kita dapat mengeksekusi suatu blok kode tanpa harus menulisnya berulang-ulang. Dalam fungsi juga dapat digunakan parameter default yang dapat dilihat hasilnya saat pemanggilan fungsi dengan parameter tidak lengkap. Fungsi di Python juga memungkinkan pendeteksian banyak parameter secara otomatis melalui parameter spesial *args (arguments) dan **kwargs (keyworded arguments). Dalam *args, parameter fungsi dianggap sebagai satu kesatuan list, sedangkan dalam **kwargs parameter fungsi dianggap sebagai satu kesatuan dictionary.

```
def kelulusan(nama, nilai):
    if nilai >= 50:
        return f"{nama} lulus dengan nilai {nilai}"
    else:
        return f"{nama} mengulang dengan nilai {nilai}"

list_mhs = [["Ahmad", 80], ["Sam", 40], ["Opick", 70]]

for i in list_mhs:
    print(kelulusan(i[0], i[1]))
```

MODUL 2

2.1 Kelas

Kelas atau class pada python bisa kita katakan sebagai sebuah blueprint (cetakan) dari objek (atau instance) yang ingin kita buat. Kelas berisi dan mendefinisikan atribut/properti dan metode untuk objeknya nanti. Satu buah kelas dapat membuat banyak objek dan kelas sendiri tidak bisa langsung digunakan, harus diimplementasikan menjadi sebuah objek dulu, dapat disebut Instansiasi. contohnya: kelas Mobil, kelas Manusia, dan kelas Kucing. Untuk membuat kelas, gunakan kata kunci class diikuti oleh nama kelas tersebut dan tanda titik dua. Contoh:

```
class Orang:
    jumlah_mata = 2
    jumlah_hidung = 1

    def __init(self, nama, pekerjaan, tahun_lahir):
        self.nama = nama
        self.pekerjaan = pekerjaan
        self.tahun_lahir = tahun_lahir

orang1 = Orang("Enrico", "Mahasiswa", 2000)
orang2 = Orang("Andi", "Direktur", 1980)

print(orang1)
print(orang2)
```

Dapat dilihat diatas terdapat __init__(), method tersebut adalah konstruktor untuk membuat kelas orang. Kelas mungkin merupakan cara termudah bagi kita untuk mengorganisasikan data ketika melakukan komputasi ilmiah. Meskipun demikian hal ini bukan tanpa kelemahan, terkadang dengan mengorganisasikan data kita ke dalam kelas, program yang kita buat menjadi lambat ketika dijalankan

2.1.1 Atribut

Dalam suatu kelas, umumnya dideklarasikan sebuah variabel yang akan disebut sebagai sebuah atribut. Karena dideklarasikan di dalam sebuah kelas, maka setiap objek yang dibentuk dari kelas tersebut juga akan memiliki atribut yang dimiliki oleh kelas tersebut. Terdapat 2 jenis atribut, yaitu atribut kelas dan atribut objek. Atribut kelas merupakan sifat yang dimiliki oleh sebuah kelas dan juga akan dimiliki oleh setiap objek. Sedangkan atribut objek adalah sebuah atribut dari masing - masing objek.

```
#atribut kelas
jumlah_mata = 2
jumlah_hidung = 1

#atribut objek
def __init(self, nama, pekerjaan, tahun_lahir):
    self.nama = nama
    self.pekerjaan = pekerjaan
    self.tahun_lahir = tahun_lahir
```

2.1.2 Method

Method adalah suatu fungsi yang terdapat di dalam kelas. Sama halnya seperti atribut, semua objek yang dibuat menggunakan kelas yang sama akan memiliki method yang sama pula. Method dapat diibaratkan sebagai sebuah aktivitas/proses yang dapat dilakukan oleh sebuah objek.

```
def berjalan_kedepan(self):
    print("Melangkah ke depan")
```

2.2 Objek

Objek adalah sesuatu yang "mewakili" kelas. Objek disini berfungsi sebagai pengganti pemanggilan sebuah kelas, maka sebuah objek hanya dapat mewakili sebuah kelas saja. Untuk membuat sebuah objek yang mewakili sebuah kelas, kita dapat membuatnya dengan cara memanggil nama dari kelas yang diinginkan, ditambah dengan tanda kurung ( )

```
orang1 = Orang("Enrico", "Mahasiswa", 2000)
orang2 = Orang("Andi", "Direktur", 1980)
```

2.3 Magic Method

Magic method adalah metode yang diawali dan diakhiri dengan double underscore (dunder). Method ini tidak dipanggil secara langsung, tapi dipanggil sistem secara internal ketika melakukan sesuatu seperti menggunakan operator tambah (__add__), membuat objek (__init__), dan lain-lain. Tujuan utama dari penggunaan magic method adalah untuk mengubah sifat (behavior) bawaan dari suatu objek. Untuk melihat apa saja magic method bawaan python pada class int, gunakan sintaks dir(int).

```
>>> dir(int)
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '__delattr__',
'__dir__', '__divmod__', '__doc__', '__eq__', '__float__', '__floor__', '__floordiv__',
'__format__', '__ge__', '__getattribute__', '__getnewargs__', '__gt__', '__hash__',
'__index__', '__init__', '__init_subclass__', '__int__', '__invert__', '__le__',
'__lshift__', '__lt__', '__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__or__',
'__pos__', '__pow__', '__radd__', '__rand__', '__rdivmod__', '__reduce__', '__reduce_ex__',
'__repr__', '__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__', '__ror__', '__round__',
'__rpow__', '__rrshift__', '__rshift__', '__rsub__', '__rtruediv__', '__rxor__',
'__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__truediv__',
'__trunc__', '__xor__', 'bit_length', 'conjugate', 'denominator', 'from_bytes', 'imag',
'numerator', 'real', 'to_bytes']
```

Salah satu contoh magic method yaitu __add__(). Method ini ditambahkan agar user dapat melakukan operasi penambahan (+) secara langsung pada objek, tanpa mengakses atribut isi objek (dalam hal ini yaitu self.angka).

2.4 Konstruktor

Konstruktor adalah method yang "pasti" dijalankan secara otomatis pada saat sebuah objek dibuat untuk mewakili kelas tersebut. Seperti dengan method lain, konstruktor dapat melakukan operasi seperti melakukan print. Selain operasi method dasar, konstruktor dapat menerima argumen yang diberikan ketika objek dibuat. Argumen ini akan diproses di dalam kelas nantinya. Contoh :

```
class Panggil_kelas:

    def __init__(self, nama, pekerjaan, tahun_lahir):
        self.nama = nama
        self.pekerjaan = pekerjaan


    def panggilnama(self):
        print("Nama : ", self.nama)

    def panggilpekerjaan(self):
        print("Pekerjaan : ", self.pekerjaan)

orang1 = Panggil_kelas("Enrico", "Mahasiswa")

orang1.panggilnama()
orang1.panggilpekerjaan()
```

2.5 Destruktor

Destruktor adalah fungsi yang dipanggil ketika user menghapus objek. Fungsi ini bekerja secara otomatis, jadi tidak perlu dilakukan pemanggilan. Tujuan adanya destruktor adalah melakukan final cleaning up atau "bersih-bersih" terakhir sebelum sebuah objek benar-benar dihapus dari memori.

```
def __del__(self):
    print("Mahasiswa dengan nama", self.nama, "dihapus")
```

2.6 Setter dan Getter

Setter dan getter digunakan untuk melakukan enkapsulasi agar tidak terjadi perubahan data secara tidak sengaja. Setter adalah method yang digunakan untuk menetapkan nilai suatu atribut khususnya atribut private dan protected (akan diajarkan lebih jelas pada materi minggu selanjutnya), sedangkan getter digunakan untuk mengambil nilai.

```
class Mahasiswa:
    def __init__(self, semester):
        self._semester = semester

    #Getter Method
    def get_semester(self):
        return self._semester

    #Setter Method
    def set_semester(self, x):
        return self._semester = x

maha = Mahasiswa()
maha.set_semester(4)
print(maha.get_semester())
print(maha._semester)
```

2.7 Decorator

Selain menggunakan fungsi setter dan getter tambahan seperti pada contoh sebelumnya, dalam Python kita juga dapat memanfaatkan property decorator untuk mendapatkan hasil serupa. Bedanya, melalui property decorator ini kita tidak perlu membuat fungsi lagi dengan nama yang berbeda-beda (cukup menggunakan 1 buah nama/variabel). Contoh penggunaan decorator:

```
class Mahasiswa:
    def __init__(self, nama, umur = 21):
        self.__nama = nama
        self.__umur = umur

    @property
    def umur(self):
        return self.__umur

    @umur.setter
    def umur(self, x):
        self.__umur = x

Juna = Mahasiswa("Juna")
print(Juna.umur)
Juna.umur = 22
print(Juna.umur)
```

MODUL 3

3.1 Abstraksi

Abstraksi adalah konsep OOP dimana model yang dibuat hanya memperlihatkan atribut yang esensial dan menyembunyikan detail-detail yang tidak penting dari user. gunanya untuk mengurangi kompleksitas. User mengetahui apa yang objek lakukan, tapi tidak tau mekanisme yang terjadi di belakang layar bagaimana. Contohnya ketika mengendarai mobil, user mengetahui bagaimana menyalakan mobil, menjalankan, menghentikan, dll, tetapi tidak mengetahui mekanisme apa yang terjadi pada mobil ketika mendapat perintah di atas. Ketika mendefinisikan kelas, sebenarnya sedang membuat Abstrak dari suatu Objek. Kelas merupakan bentuk abstrak atau cetak biru (blueprint) dari suatu objek nyata. Wujud nyata suatu kelas dinamakan instance (Objek).

```
class Sepeda:
    def __init__(self):
        self.nama = "Polygon"
        self.__max_kecepatan = 2
        self.__kecepatan= 0

    def get_kecepatan(self):
        return self.__kecepatan
```

3.2 Enkapsulasi

Enkapsulasi adalah sebuah metode yang digunakan untuk mengatur struktur kelas dengan cara menyembunyikan alur kerja dari kelas tersebut, yang dimaksud dengan struktur kelas tersebut adalah property dan method. Dengan konsep ini, kita dapat "menyembunyikan" property dan method dari suatu kelas agar hanya property dan method tertentu saja yang dapat diakses dari luar kelas. Dengan menghalangi akses dari luar kelas tersebut, elemen penting yang terdapat dalam kelas dapat lebih terjaga, dan menghindari kesalahan jika elemen tersebut diubah secara tidak sengaja. Abstraksi adalah cara untuk menyembunyikan informasi yang tidak dibutuhkan, sedangkan enkapsulasi adalah cara menyembunyikan atribut suatu entitas serta metode untuk melindungi informasi dari luar. Untuk membatasi hak akses terhadap property dan method dalam suatu kelas, terdapat 3 jenis access modifier yang terdapat dalam python, yaitu public access, protected access, dan private access.

3.2.1 Public Access Modifier

Pada umumnya ketika kita mendeklarasikan suatu variabel atau method, maka itulah public access modifier. Setiap class, variable dan method yang dibuat secara default merupakan public.

```
#Public Access Modifier
class Mahasiswa:
    global_jumlah_variable = 0

    def __init__(self, nama, semester):
        self.nama = nama
        self.semester = semester
        Mahasiswa.global_jumlah_variable += 1

    def printjumlah(self):
        print("Jumlah Mahasiswa : ", Mahasiswa.global_jumlah_variable)

    def printprofil(self):
        print("Nama : ", self.nama)
        print("Semester : ", self.semester)
```

3.2.2 Protected Access Modifier

Jika suatu variabel dan method dideklarasikan secara protected, maka variable dan method tersebut hanya dapat diakses oleh kelas turunan darinya. Cara mendeklarasikannya dengan menambahkan 1 underscore (_) sebelum variable atau method.

```
#Protected Access Modifier
class Mobil:
    _merk = None
    _warna = None

    def __init__(self, nama, warna, jenis, tipe):
        self._nama = nama
        self._warna = warna
        self.__jenis = jenis
        self.__tipe = tipe

    def _merkwarna(self):
        print("Merk Mobil : ", self._nama)
        print("Warna Mobil : ", self._warna)

    def _jenistipe(self):
        print("Jenis Mobil : ", self.__jenis)
        print("Tipe Mobil : ", self.__tipe)

mobil = Mobil("Toyota", "Hitam", "SUV", "Fortuner")
mobil._merkwarna()
mobil._jenistipe()
```

3.2.3 Privated Access Modifier

Jika suatu variabel dan method dideklarasikan secara private, maka variable dan method tersebut hanya dapat diakses di dalam kelas itu sendiri, private access merupakan yang 23 paling aman. Dalam mendeklarasikannya, hanya perlu menambahkan double underscore ( ) sebelum nama variable dan methodnya.

```
#Privated Access Modifier
class Mobil:
    __jenis = None
    __tipe = None

    def __init__(self, jenis, tipe):
        self.__jenis = jenis
        self.__tipe = tipe

    def _merkwarna(self):
        print("Merk Mobil : ", self._nama)
        print("Warna Mobil : ", self._warna)

    def _jenistipe(self):
        print("Jenis Mobil : ", self.__jenis)
        print("Tipe Mobil : ", self.__tipe)

mobil = Mobil("Toyota", "Hitam", "SUV", "Fortuner")
mobil._merkwarna()
mobil._jenistipe()
```

MODUL 4

4.1 Inheritance

Sintaks Dasar :

```
class Parent:
    pass

class Child(Parent):
    pass
```

Inheritance adalah salah satu konsep dasar dari Pemrograman Berbasis Objek (OOP). Pada inheritance, kita dapat menurunkan kelas dari kelas lain untuk hirarki kelas yang saling berbagi atribut dan metode. Contohnya terdapat base class “Animal” dan terdapat class “Horse” yang merupakan turunan dari class “Animal”. ini berarti class “Horse” memiliki atribut dan method yang sama dengan class “Animal”. Dan objek “Horse” dapat menggantikan objek “Animal” dalam aplikasi.

```
class Manusia:
    def __init__(self, nama, umur):
        self.nama = nama
        self.umur = umur

    def profile(self):
        return f"Nama : {self.nama}, Umur : {self.umur}"

class Mahasiswa(Manusia):
    pass

mhs = Mahasiswa("Meazza", 20)
mhs.profile()
```

4.1.1 Inheritance Identik

Inheritance identik merupakan pewarisan yang menambahkan constructor pada class child sehingga class child memiliki constructornya sendiri tanpa menghilangkan constructor pada class parentnya. Metode ini ditandai dengan adanya class child yang menggunakan constructor dan menggunakan kata kunci super().

```
class Binatang: #Parent Class
    def __init__(self, nama, jk, jenis):
        self.nama = nama
        self.jk = jk
        self.jenis = jenis

    def __str__(self): #Magic Method
        return f"Nama: {self.nama}\nJenis Kelamin: {self.jk}\nJenis: {self.jenis}"

class Kucing(Binatang):
    def __init__(self, nama, jk, jenis, warna):
        super().__init__(nama, jk, jenis)
        self.warna = warna
```

4.1.2 Inheritance Tidak Identik

Pada child class dapat ditambahkan beberapa fitur tambahan baik atribut maupun method sehingga child class tidak identik dengan parent class.

```python
class PersegiPanjang():
    def __init__(self, p, l):
        self.panjang = p
        self.lebar = l

    def luas(self):
        return self.panjang * self.lebar

class Balok(PersegiPanjang):
    def __init__(self, p, l, t):
        super().__init__(p, l)
        self.tinggi = t

    def volume(self):
        return self.luas() * self.tinggi
```

4.2 Polymorphisme

Polymorphism berarti banyak (poly) dan bentuk (morphism), dalam Pemrograman Berbasis Objek konsep ini memungkinkan digunakannya suatu interface yang sama untuk memerintah objek agar melakukan aksi atau tindakan yang mungkin secara prinsip sama namun secara proses berbeda. Polymorphism merupakan kemampuan suatu method untuk bekerja dengan lebih dari satu tipe argumen. Pada bahasa lain (khususnya C++), konsep ini sering disebut dengan method overloading. Pada dasarnya, Python tidak menangani hal ini secara khusus. Hal ini disebabkan karena Python merupakan suatu bahasa pemrograman yang bersifat duck typing(dynamic typing).

```python
class Timun():
    def jenis(self):
        print("Sayuran")
    def warna(self):
        print("Hijau")

class Anggur():
    def jenis(self):
        print("Buah")
    def warna(self):
        print("Ungu")

def fungsi(obj):
    obj.jenis()
    obj.warna()

obj1 = Timun()
obj2 = Anggur()
fungsi(obj1)
fungsi(obj2)
```

4.3 Override

Pada konsep OOP di python kita dapat menimpa suatu metode yang ada pada parent class dengan mendefinisikan kembali method dengan nama yang sama pada child class. Dengan begitu maka method yang ada parent class tidak berlaku dan yang akan dijalankan adalah method yang terdapat di child class.

```
class Bangundatar():
    def cetak(self):
        print("ini bangun datar")

class Segitiga(Bangundatar):
    def cetak(self):
        print("ini segitiga")

s1 = Segitiga()
s1.cetak
```

4.4 Overloading

Metode overloading mengizinkan sebuah class untuk memiliki sekumpulan fungsi dengan nama yang sama dan argumen yang berbeda. Akan tetapi, Python tidak mengizinkan pendeklarasian fungsi (baik pada class ataupun tidak) dengan nama yang sama. Hal itu menyebabkan implementasi overloading pada python menjadi "tricky". Secara umum overloading memiliki beberapa signature, yaitu jumlah argumen, tipe argumen, tipe keluaran dan urutan argumen. Akan tetapi, seperti yang telah dijelaskan python tidak menangani overloading secara khusus. Karena python adalah bahasa pemrograman yang bersifat duck typing (dynamic typing), overloading secara tidak langsung dapat diterapkan.

```
class Bangundatar():
    def __init__(self, sisi):
        self.sisi = sisi

    def luas(self):
        print("ini luas bangun datar")

class Segitiga:
    def __init__(self, jenis):
        self.jenis = jenis

    def luas(self):
        print("ini luas segitiga")

def luas(obj):
    obj.luas()

bgndtr = Bangundatar("kanan")
sgtg = Segitiga("sembarang")
luas(bgndtr)
luas(sgtg)
```

4.5 Multiple Inheritance

Python mendukung pewarisan ke banyak kelas. Kelas dapat mewarisi dari banyak orang tua. Bentuk syntax multiple inheritance adalah sebagai berikut : Gambar 4.1.2.1 Sintaks Multiple Inheritance Dimana class Derived merupakan kelas yang berasal dari class Base1 dan class Base2. Kita juga dapat mewarisi dari kelas turunan atau disebut warisan bertingkat. Pewarisan ini dapat dilakukan hingga ke dalaman berapa pun. Dalam kelas turunan dari kelas dasar dan turunannya dapat diwarisi ke dalam kelas turunan yang baru. Contoh :

```
class Base1:
    pass

class Base2:
    pass

class Derived(Base1, Base2):
    pass
```

```
class Base:
    pass

class Derived1(Base):
    pass

class Derived2(Derived1):
    pass
```

4.6 Method Resolution Order

MRO adalah urutan pencarian metode dalam hirarki class. Hal ini terutama berguna dalam multiple inheritance. Urutan MRO dalam python yaitu bawah-atas dan kiri-kanan. Artinya, method dicari pertama kali di kelas objek. jika tidak ditemukan, pencarian berlanjut ke super class. Jika terdapat banyak superclass (multiple inheritance), pencarian dilakukan di kelas yang paling kiri dan dilanjutkan ke kelas sebelah kanan.

```
class X:
    pass

class Y:
    pass

class Z:
    pass

class A(X,Y):
    pass

class B(X,Y):
    pass

class C(A,B,Z):
    pass
```

4.7 Dynamic Cast

Dynamic cast atau type conversion adalah proses mengubah nilai dari satu tipe data ke tipe data lainnya seperti dari string ke int atau sebaliknya. Ada 2 tipe konversi yaitu implisit dan eksplisit.

4.7.1 Implisit

Python secara otomatis mengkonversikan tipe data ke tipe data lainnya tanpa ada campur tangan pengguna.

```python
num_int = 123
num_flo = 1.23

num_new = num_int + num_flo

print("tipe data num int: ", type(num_int))
print("tipe data num flo: ", type(num_flo))

print("nilai num_new: ", num_new)
print("tipe data num_new: ", type(num_new))
```

4.7.2 Eksplisit

Pengguna mengubah tipe data sebuah objek ke tipe data lainnya dengan fungsi yang sudah ada dalam python seperti int(), float(), dan str(). dapat berisiko terjadinya kehilangan data.

```python
num_int = 123
num_str = "1.23"

print("tipe data num int: ", type(num_int))
print("tipe data num str sebelum typecasting: ", type(num_str))

num_str = int(num_str)
print("tipe data num str setelah typecasting: ", type(num_str))

num_sum = num_int + num_str

print("nilai num_sum: ", num_sum)
print("tipe data num_sum: ", type(num_sum))
```

4.8 Casting

4.8.1 Downcasting

Parent class mengakses atribut yang ada pada kelas bawah (child class).

```python
class Manusia:
    def __init__(self, namadepan, namabelakang):
        self.namadepan = namadepan
        self.namabelakang = namabelakang

    def biodata(self):
        print(f"{self.namadepan} {self.namabelakang} ({self.profesi})")

class Pekerja(Manusia):
    def __init__(self, namadepan, namabelakang, profesi):
        super().__init__(namadepan, namabelakang)
        self.profesi = profesi

Ahmad = Pekerja("Ahmad", "Junaedi", "Teknisi")
Ahmad.biodata()
```

4.8.2 Upcasting

Child class mengakses atribut yang ada pada kelas atas (parent class).

```
class Manusia:
    namabelakang = "Junaedi"

    def __init__(self, namadepan, namabelakang):
        self.namadepan = namadepan
        self.namabelakang = namabelakang

    def biodata(self):
        print(f"{self.namadepan} {self.namabelakang} ({self.profesi})")

class Pekerja(Manusia):
    def __init__(self, namadepan, namabelakang, profesi):
        super().__init__(namadepan, namabelakang)
        self.profesi = profesi

    def biodata(self):
        print(f"{self.namadepan} {super().namabelakang} ({self.profesi})")

Ahmad = Pekerja("Ahmad", "Junaedi", "Teknisi")
Ahmad.biodata()
```

4.8.3 Typecasting

Konversi tipe kelas agar memiliki sifat/perilaku tertentu yang secara default tidak dimiliki kelas tersebut. Karena Python merupakan bahasa pemrograman berorientasi objek, maka semua variabel atau instansi di Python pada dasarnya merupakan objek (kelas) yang sifat/perilakunya dapat dimanipulasi jika dibutuhkan (umumnya melalui magic method)

```
class Mahasiswa:
    def __init__(self, nama, nim, matkul):
        self.__nama = nama
        self.__nim = nim
        self.__matkul = matkul

    def __str__(self):
        return f"{self.__nama} ({self.__nim}) merupakan mahasiswa kelas {self.__matkul}"

    def __int__(self):
        return self.__nim

Mario = Mahasiswa("Mario", 121140, "PBO RB")
print(Mario)
print(int(Mario) == 121140)
```

DAFTAR PUSTAKA

Modul 1 Praktikum Pemrograman Berbasis Objek

Modul 2 Praktikum Pemrograman Berbasis Objek

Modul 3 Praktikum Pemrograman Berbasis Objek

Modul 4 Praktikum Pemrograman Berbasis Objek