RESUME MODUL 1, 2, 3, 4 PRAKTIKUM
PEMROGRAMAN BERORIENTASI OBJEK

Disusun oleh :

Qaisya Dwi Aryana

121140063

RB

PROGAM STUDI TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI SUMATERA

2023

Pengertian Python

Python adalah bahasa pemrograman tingkat tinggi yang dikembangkan oleh Guido van Rossum pada tahun 1991. Python didesain untuk menyederhanakan proses pemrograman dengan sintaks yang mudah dibaca dan dipahami oleh manusia. Dalam pemrograman Python, kita bisa membuat berbagai macam aplikasi seperti aplikasi desktop, aplikasi web, aplikasi mobile, game, kecerdasan buatan, ilmu data, dan lainnya. Python juga mendukung berbagai paradigma pemrograman, salah satunya pemrograman berorientasi objek. Python tidak menggunakan kurung kurawal({}) atau keyword (ex. start, begin, end), sebagai gantinya menggunakan spasi(white space) untuk memisahkan blok-blok kode. Selain itu, Python juga dikenal sebagai bahasa pemrograman yang mudah dipelajari, sehingga cocok bagi pemula yang ingin belajar pemrograman.

Dasar Pemprograman Python

- Sintaks Dasar

Statement

Statement dalam bahasa pemrograman Python adalah instruksi atau perintah yang digunakan untuk melakukan tindakan atau operasi tertentu. Statement dalam Python diakhiri dengan tanda titik dua (:) dan dapat diikuti dengan blok kode yang terdiri dari satu atau beberapa baris kode yang diindentasi. Setiap statement dalam Python memiliki arti dan fungsi masing-masing tergantung dari jenisnya. Sebagai contoh, statement "print" digunakan untuk menampilkan teks atau nilai dari sebuah variabel ke layar, sedangkan statement "if" digunakan untuk melakukan percabangan pada program.

Baris dan Indentasi

Setiap perintah atau instruksi dalam Python harus ditulis dalam satu baris yang berbeda. Namun, jika perlu, baris bisa dibagi menjadi beberapa bagian menggunakan karakter. Sedangkan indentasi digunakan untuk menunjukkan blok kode yang saling terkait. Setiap baris dalam sebuah blok harus diberi indentasi yang sama, biasanya menggunakan empat spasi atau satu tab.

```
1  x = -10
2
3  if x < 0:
4      print("x adalah bilangan negatif")
5  else:
6      print("x adalah bilangan positif atau nol")
7
```

Pada contoh diatas, diketahui blok kode ”if” dan ”else” memiliki indentasi yang sama yakni 1 tab, seingga hal itu memberitahu bahwa kedua blok tersebut adalah bagian dari percabangan yang sama.

- Variabel dan Tipe Data Primitif
  Variabel merupakan lokasi penyimpanan yang berguna untuk menyimpan suatu data atau suatu nilai. Serta memiliki tipe data sebagai berikut:

Tipe data	Jenis	Contoh nilai
Integer ’int’	Bilangan bulat	1, 2, 3, -4, -5, ..dst
Float ’float’	Bilangan pecahan (bil real)	3.14, 2.5, -0.5, ..dst
Boolean ’bool’	Nilai kebenaran	’True’ atau ’False’
String ’str’	Teks atau karakter	”Hello World”, ”abc” ..dst

- Operator

  Python memeiliki beberapa operator, yakni:

  1) Operator Aritmatika

     Operator Aritmatika digunakan untuk melakukan operasi matematika pada nilai numerik. Contohnya:

Operator	Fungsi
+	Penjumlahan
-	Pengurangan
*	Perkalian
/	Pembagian
%	Modulus, sisa pembagian
**	Pangkat

  2) Operator Perbandingan

     Operator Perbandingan: digunakan untuk membandingkan nilai dan menghasilkan nilai boolean True atau False. Contohnya:

Operator	Fungsi
>	Lebih besar dari
<	Lebih kecil dari
>=	Lebih besar atau sama dengan
<=	Lebih kecil atau sama dengan
==	Sama dengan
!=	Tidak sama dengan

  3) Operator Logika

     Operator Logika: digunakan untuk mengkombinasikan nilai boolean True atau False. Contohnya:

Operator	Fungsi
And	And
Or	Or
not	Negasi

  4) Operator Bitwise

     Operator Bitwise: digunakan untuk melakukan operasi bit pada nilai numerik. Contohnya:

Operator	Fungsi
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT

<<	Left Shift
>>	Right Shift

- Tipe Data Bentukan
  Selain tipe data dasar seperti integer, float, string, dan boolean, Python juga memiliki tipe data bentukan (data structures) yang lebih kompleks dan lebih fleksibel untuk menyimpan data. Tipe data bentukan ini memungkinkan kita untuk menyimpan beberapa nilai atau objek dalam satu variabel, sehingga memudahkan dalam pengelolaan data. Berikut adalah beberapa tipe data bentukan yang ada pada Python:
  1. List: digunakan untuk menyimpan kumpulan data atau nilai yang berbeda-beda. List ditandai dengan kurung siku [], dan nilai-nilai pada list dipisahkan dengan koma.
  2. Tuple: mirip dengan list, namun tuple bersifat tidak dapat diubah setelah dibuat (immutable). Tuple ditandai dengan tanda kurung biasa (), dan nilai-nilai pada tuple dipisahkan dengan koma.
  3. Set: adalah tipe data bentukan yang digunakan untuk menyimpan kumpulan nilai yang unik dan tidak terurut. Set ditandai dengan kurung kurawal {}, dan nilai-nilai pada set dipisahkan dengan koma.
  4. Dictionary: adalah tipe data bentukan yang digunakan untuk menyimpan pasangan key-value atau kunci-nilai. Dictionary ditandai dengan kurung kurawal {}, dan setiap pasangan key-value dipisahkan dengan koma dan dipisahkan oleh tanda titik dua.

- Percabangan
  Python memiliki beberapa pernyataan percabangan yang bisa digunakan, yaitu if, if-else, if-elif-else, dan nested if.
  1. Percabangan IF
     ```
     x = -10

     if x < 0:
         print("x adalah bilangan negatif")
     ```
  2. Percabangan IF-ELSE
     ```
     x = -10

     if x < 0:
         print("x adalah bilangan negatif")
     else:
         print("x adalah positif atau nol")
     ```
  3. Percabangan IF-ELIF-ELSE

```
x = -10

if x < 0:
    print("x adalah bilangan negatif")
elif x > 0:
    print("x adalah positif")
else:
    print("x adalah 0")
```

4. Nested IF
   Nested if adalah sebuah kondisi di mana sebuah statement if-else di dalamnya memiliki statement if-else lagi di dalamnya. Contohnya sebagai berikut:

```
n = int(input())

if(n <= 0):
    if(n==0):
        print(str(n) + " bilangan nol")
    else:
        print(str(n) + " bilangan negatif")

else:
    print(str(n) + " bilagan positif")
```

- Perulangan
  Dalam python terdapat dua jenis perulangan , yakni perulangan for dan perulangan while. Dalam implementasi nya perulangan digunakkan jika ingin mengulang sesuatu sebanyak n kali.
  1. Perulangan FOR
     Perulangan for digunakan untuk melakukan iterasi melalui sebuah urutan (seperti daftar, tuple, atau string) dan menjalankan blok kode untuk setiap elemen dalam urutan. Contoh penggunaan perulangan for:

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

  2. Perulangan WHILE
     Perulangan while digunakan untuk menjalankan blok kode selama kondisi yang ditentukan terpenuhi. Contoh penggunaan perulangan while:

```
i = 0
while i < 10:
    print(i)
    i += 1
```

Kelas

- Atribut
  Atribut adalah variabel yang didefinisikan di dalam kelas dan digunakan untuk menyimpan data atau informasi dalam objek. Atribut dapat didefinisikan secara publik atau pribadi. Atribut publik dapat diakses dari luar kelas, sedangkan atribut pribadi hanya dapat diakses di dalam kelas. Atribut dapat diakses dengan menggunakan operator titik (.). Contoh:

```
class Mobil:
    def __init__(self, warna, merk):
        self.warna = warna
        self.merk = merk

    def tampilan(self):
        print("Mobil " + self.merk + " berwarna " + self.warna)

mobil1 = Mobil("merah", "Toyota")
mobil1.tampilan()
```

- Method
  Method adalah fungsi yang didefinisikan di dalam kelas dan digunakan untuk melakukan tugas tertentu. Method dapat didefinisikan sebagai method publik atau pribadi. Method publik dapat diakses dari luar kelas, sedangkan method pribadi hanya dapat diakses di dalam kelas. Method dapat dipanggil dengan menggunakan operator titik (.). Contoh:

```
class Mobil:
    def __init__(self, warna, merk):
        self.warna = warna
        self.merk = merk

    def tampilan(self):
        print("Mobil " + self.merk + " berwarna " + self.warna)

    def gantiWarna(self, warnaBaru):
        self.warna = warnaBaru

mobil1 = Mobil("merah", "Toyota")
mobil1.tampilan()

mobil1.gantiWarna("biru")
mobil1.tampilan()
```

- Objek
  Objek adalah instansi dari kelas. Objek dibuat dengan menggunakan konstruktor kelas. Setiap objek memiliki atribut dan method yang didefinisikan dalam kelas. Objek dapat diakses dengan menggunakan operator titik (.). Contoh:

```
class Mobil:
    def __init__(self, warna, merk):
        self.warna = warna
        self.merk = merk

    def tampilan(self):
        print("Mobil " + self.merk + " berwarna " + self.warna)

mobil1 = Mobil("merah", "Toyota")
mobil1.tampilan()
```

- Magic Method
  Magic Method adalah method khusus dalam kelas yang digunakan untuk memberikan perilaku tambahan pada objek. Magic Method didefinisikan dengan nama yang diawali dan diakhiri dengan dua underscore (__). Beberapa contoh Magic Method yang umum digunakan adalah __init__, __str__, __eq__, dan __lt__. Contoh:

```
class Mobil:
    def __init__(self, warna, merk):
        self.warna = warna
        self.merk = merk

    def __str__(self):
        return "Mobil " + self.merk + " berwarna " + self.warna

mobil1 = Mobil("merah", "Toyota")
print(mobil1)
```

- Konstruktor
  Konstruktor adalah method khusus dalam kelas yang digunakan untuk membuat objek. Konstruktor didefinisikan dengan nama __init__. Konstruktor dapat menerima argumen untuk menginisialisasi atribut dalam objek. Contoh:

```
class Mobil:
    def __init__(self, warna, merk):
        self.warna = warna
        self.merk = merk

    def tampilan(self):
        print("Mobil " + self.merk + " berwarna " + self.warna)

mobil1 = Mobil("merah", "Toyota")
mobil1.tampilan()
```

- Destruktor
  Destruktor adalah method khusus dalam kelas yang digunakan untuk membersihkan objek setelah digunakan. Destruktor didefinisikan dengan nama __del__ dan secara otomatis dipanggil ketika objek dihapus atau program selesai dijalankan. Contoh:

```python
class Mobil:
    def __init__(self, warna, merk):
        self.warna = warna
        self.merk = merk

    def __del__(self):
        print("Objek Mobil dihapus")

    def tampilan(self):
        print("Mobil " + self.merk + " berwarna " + self.warna)

mobil1 = Mobil("merah", "Toyota")
mobil1.tampilan()

del mobil1
```

- Setter dan Getter
  Setter dan Getter adalah method khusus dalam kelas yang digunakan untuk mengatur (set) dan mendapatkan (get) nilai atribut dalam objek. Setter digunakan untuk mengubah nilai atribut, sedangkan Getter digunakan untuk membaca nilai atribut. Setter dan Getter didefinisikan dengan menggunakan dekorator @property dan @nama.setter. Contoh:

```python
class Mobil:
    def __init__(self, warna, merk):
        self.warna = warna
        self.merk = merk

    @property
    def warnaMobil(self):
        return self.warna

    @warnaMobil.setter
    def warnaMobil(self, warnaBaru):
        self.warna = warnaBaru

    def tampilan(self):
        print("Mobil " + self.merk + " berwarna " + self.warna)

mobil1 = Mobil("merah", "Toyota")
mobil1.tampilan()

mobil1.warnaMobil = "biru"
print(mobil1.warnaMobil)
```

- Decorator
  Decorator adalah fungsi yang digunakan untuk mengubah perilaku fungsi atau method dalam Python. Decorator dapat digunakan untuk menambahkan fungsionalitas baru pada fungsi atau method tanpa mengubah kode asli. Decorator didefinisikan dengan menggunakan tanda @ diikuti dengan nama decorator. Contoh:

```python
def uppercase_decorator(function):
    def wrapper():
        func = function()
        make_uppercase = func.upper()
        return make_uppercase
    return wrapper

@uppercase_decorator
def greeting():
    return "Hello World!"

print(greeting())
```

Abstraksi

Abstraksi pada Python dapat diimplementasikan dengan menggunakan konsep kelas dan objek. Dalam konsep OOP, kelas merupakan sebuah blueprint atau cetak biru untuk membuat objek, sedangkan objek adalah sebuah instance atau wujud nyata dari kelas tersebut. Dengan menggunakan konsep ini, kita dapat membuat sebuah model atau representasi yang menyembunyikan detail-detail yang tidak penting dari pengguna, sehingga kompleksitas program dapat dikurangi.

```python
class Car:
  def start(self):
    # kode untuk menyalakan mobil

  def stop(self):
    # kode untuk menghentikan mobil

  def drive(self):
    # kode untuk menggerakkan mobil

my_car = Car()
my_car.start()
my_car.drive()
my_car.stop()
```

Enkapsulasi

Enkapsulasi adalah konsep yang terkait dengan abstraksi di mana kita membatasi akses langsung ke data dan metode dalam suatu objek. Dalam Python, ada tiga jenis enkapsulasi yang umum digunakan: public, protected, dan private.

1. Public Enkapsulasi
   Variabel atau metode yang didefinisikan sebagai public dapat diakses dari mana saja di dalam program. Untuk membuat variabel atau metode menjadi public, cukup deklarasikan tanpa menggunakan tanda garis bawah (_).

```python
class Mahasiswa:
    def __init__(self, nama, npm):
        self.nama = nama
        self.npm = npm

    def display_data(self):
        print("Nama: ", self.nama)
        print("NPM: ", self.npm)

mhs = Mahasiswa("Andi", "12345")
mhs.display_data()
```

2. Protected Enkapsulasi
   Variabel atau metode yang didefinisikan sebagai protected hanya dapat diakses dari kelas itu sendiri dan subkelasnya. Untuk membuat variabel atau metode menjadi protected, gunakan tanda garis bawah tunggal (_) sebelum nama variabel atau metode.

```python
class Mahasiswa:
    def __init__(self, nama, npm):
        self._nama = nama
        self._npm = npm

    def display_data(self):
        print("Nama: ", self._nama)
        print("NPM: ", self._npm)

class MahasiswaBaru(Mahasiswa):
    def __init__(self, nama, npm, jurusan):
        super().__init__(nama, npm)
        self._jurusan = jurusan

    def display_data(self):
        super().display_data()
        print("Jurusan: ", self._jurusan)

mhs = MahasiswaBaru("Andi", "12345", "Teknik Informatika")
mhs.display_data()
```

3. Private Enkapsulasi
   Variabel atau metode yang didefinisikan sebagai private hanya dapat diakses dari kelas itu sendiri. Untuk membuat variabel atau metode menjadi private, gunakan tanda garis bawah ganda (__).

```python
class Mahasiswa:
    def __init__(self, nama, npm):
        self.__nama = nama
        self.__npm = npm

    def display_data(self):
        print("Nama: ", self.__nama)
        print("NPM: ", self.__npm)

class MahasiswaBaru(Mahasiswa):
    def __init__(self, nama, npm, jurusan):
        super().__init__(nama, npm)
        self.__jurusan = jurusan

    def display_data(self):
        super().display_data()
        print("Jurusan: ", self.__jurusan)

mhs = MahasiswaBaru("Andi", "12345", "Teknik Informatika")
mhs.display_data()
```

Inheritance

Inheritance adalah konsep dalam pemrograman berorientasi objek (OOP) di mana sebuah kelas dapat mewarisi sifat dan perilaku dari kelas lain yang disebut sebagai kelas induk atau super class. Kelas yang mewarisi sifat dan perilaku dari kelas induk disebut kelas turunan atau subclass. Dengan inheritance, kita dapat menghindari duplikasi kode dan mempermudah pemeliharaan kode.

Dalam Python, sebuah class dapat mewarisi sifat dan perilaku dari class lain dengan menggunakan sintaksis berikut:

```
class ClassAnak(ClassInduk):
    # kode class anak
```

Dalam contoh di atas, ClassAnak merupakan class anak yang akan mewarisi sifat dan perilaku dari ClassInduk. Setelah inheritance terjadi, ClassAnak akan memiliki semua atribut dan method yang dimiliki oleh ClassInduk, dan kita dapat menambahkan atribut dan method baru ke dalam ClassAnak sesuai kebutuhan.

Polymorphism

Polymorphism adalah konsep dalam OOP yang memungkinkan objek untuk memiliki banyak bentuk atau perilaku. Ada dua jenis polimorfisme dalam OOP: polimorfisme compile-time (atau overload) dan polimorfisme run-time (atau overriding). Polimorfisme compile-time terjadi saat kita memiliki banyak fungsi dengan nama yang sama tetapi berbeda dalam parameter atau tipe data yang diterima. Polimorfisme run-time terjadi saat kita memiliki banyak kelas turunan dengan metode yang sama dari kelas induknya tetapi dapat memiliki perilaku yang berbeda.

Override/Overriding

Override adalah konsep dalam OOP yang memungkinkan kelas turunan untuk mengubah perilaku metode yang didefinisikan dalam kelas induk. Jika kelas turunan mendefinisikan ulang metode yang sama seperti yang didefinisikan di kelas induk, maka kelas turunan dianggap meng-override metode kelas induk.

```
class Hewan:
    def bersuara(self):
        print("Hewan ini bersuara")

class Kucing(Hewan):
    def bersuara(self):
        print("Meow")

class Anjing(Hewan):
    def bersuara(self):
        print("Guk guk")
```

Overloading

Overloading adalah konsep dalam OOP yang memungkinkan kita untuk memiliki banyak fungsi dengan nama yang sama tetapi berbeda dalam parameter atau tipe data yang diterima. Ketika kita memanggil fungsi dengan nama yang sama, kompiler akan memilih fungsi yang cocok dengan parameter atau tipe data yang diterima.

```python
class Matematika:
    def hitung(self, *args):
        if len(args) == 1:
            return args[0] ** 2
        elif len(args) == 2:
            return args[0] * args[1]
        elif len(args) == 3:
            return args[0] + args[1] + args[2]

mat = Matematika()
print(mat.hitung(5))
print(mat.hitung(5, 6))
print(mat.hitung(1, 2, 3))
```

Multiple Inheritance

Multiple inheritance adalah konsep dalam OOP di mana sebuah kelas dapat mewarisi sifat dan perilaku dari lebih dari satu kelas induk atau super class. Ini memungkinkan kita untuk membuat kelas turunan yang memiliki beberapa fitur dari kelas-kelas lain.

```python
class Hewan:
    def __init__(self, nama):
        self.nama = nama

    def bersuara(self):
        pass

class Herbivora:
    def makan_tumbuhan(self):
        print("Makan tumbuhan")

class Karnivora:
    def makan_daging(self):
        print("Makan daging")

class Beruang(Hewan, Herbivora, Karnivora):
    def bersuara(self):
        print("Roaarr!")

beruang1 = Beruang("Bambang")
beruang1.makan_tumbuhan() # Output: "Makan tumbuhan"
beruang1.makan_daging() # Output: "Makan daging"
beruang1.bersuara() # Output: "Roaarr!"
print(beruang1.nama) # Output: "Bambang"
```

Method Resolution Order di Python

Method Resolution Order (MRO) adalah urutan yang digunakan Python untuk mencari metode di kelas turunan dalam hierarki kelas. Python menggunakan algoritma C3 untuk menentukan urutan metode kelas turunan.

```
D -> A -> B -> C -> E -> object
```

Artinya, ketika kita memanggil method foo pada objek dari class D, maka Python akan mencari method foo pada class D terlebih dahulu. Jika tidak ditemukan, maka Python akan mencari pada class A, dan seterusnya sesuai dengan urutan pengecekan MRO hingga ditemukan method yang sesuai atau sampai mencapai class object yang merupakan class induk dari semua class di Python. Untuk melihat urutan pengecekan MRO dari suatu class pada Python, dapat digunakan method __mro__.

Dynamic Cast

Dynamic cast tidak termasuk dalam fitur bawaan pada bahasa pemrograman Python. Sebagai gantinya, Python menyediakan operator isinstance() dan issubclass() yang dapat digunakan untuk memeriksa apakah sebuah objek atau sebuah class adalah instance atau subclass dari class tertentu. Operator isinstance() digunakan untuk memeriksa apakah sebuah objek adalah instance dari class tertentu. Contohnya sebagai berikut:

```python
class Hewan:
    def bersuara(self):
        print("Bunyi hewan")

class Kucing(Hewan):
    def bersuara(self):
        print("Meow")

class Anjing(Hewan):
    def bersuara(self):
        print("Guk-guk")

hewan1 = Kucing()
hewan2 = Anjing()

print(isinstance(hewan1, Kucing)) # Output: True
print(isinstance(hewan2, Kucing)) # Output: False
```

Operator issubclass() digunakan untuk memeriksa apakah sebuah class adalah subclass dari class tertentu. Contohnya sebagai berikut:

```python
class Hewan:
    def bersuara(self):
        print("Bunyi hewan")

class Kucing(Hewan):
    def bersuara(self):
        print("Meow")

class Anjing(Hewan):
    def bersuara(self):
        print("Guk-guk")

print(issubclass(Kucing, Hewan)) # Output: True
print(issubclass(Anjing, Kucing)) # Output: False
```

Casting

Casting pada Python adalah proses mengubah tipe data sebuah variabel menjadi tipe data yang lain. Ada beberapa tipe data yang dapat diubah ke tipe data yang lain melalui proses casting, misalnya integer ke float, float ke integer, string ke integer, dan sebagainya.