TUGAS RESUME MODUL 1-4

Disusun oleh:

Nama : Hasna Dhiya Azizah
NIM : 121140029
Kelas : Praktikum PBO RB

PROGRAM STUDI TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI SUMATERA

LAMPUNG SELATAN

2023

Modul 1

1. Pengenalan Bahasa Pemrograman Python

Bahasa pemrograman Python menjadi populer dari tahun ke tahun karena memiliki sintaks yang mudah serta didukung oleh *library* (modul) yang berlimpah. Python dapat digunakan untuk pemrograman desktop maupun mobile, CLI, GUI, web, otomatisasi, hacking, IoT, robotika, dan lain sebagainya. 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.

2. Dasar Pemrograman Python

2.1 Sintaks Dasar

a. Statement: Semua perintah yang bisa dieksekusi Python dengan akhiran baris baru

b. Baris dan Indentasi: Python menggunakan baris baru untuk menyelesaikan perintah, sementara indentasi mengacu pada spasi di awal baris kode program.

```
if 5>2:
    print('Five is greater than two!')
    print(3-2)
```

2.2 Variabel dan Tipe Data Primitif

Variabel merupakan lokasi yang berguna untuk menyimpan suatu data atau suatu nilai. Dalam mendeklarasikan suatu variabel dalam pemrograman, perlu diketahui tipe-tipe data yang berhubungan dengan variabel yang akan dideklarasikan.

```
var1=True #Boolean
angka1=4 #Integer
desimal=2.56 #Float
huruf1='Latihan Modul' #String
```

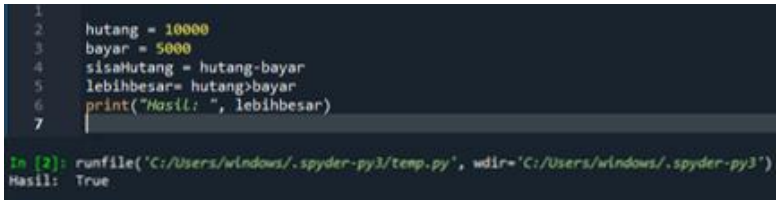
2.3 Operator

a. Operator Aritmatika: Operator yang digunakan untuk melakukan operasi matematika, seperti penjumlahan, pengurangan, perkalian, pembagian, dan sebagainya.

```
1
2   hutang = 10000
3   bayar = 5000
4   sisaHutang = hutang-bayar
5   print("Sisa hutang Anda adalah ", sisaHutang)
6

In [1]: runfile('C:/Users/windows/.spyder-py3/temp.py', wdir='C:/Users/windows/.spyder-py3')
Sisa hutang Anda adalah  5000
```

b. Operator Perbandingan: Operator yang digunakan untuk membandingkan dua buah nilai menggunakan tanda >, <, =, atau !. Hasil perbandingannya adalah True atau False tergantung kondisi.



c. Operator Penugasan: Operator yang digunakan untuk memberi nilai ke variabel. a = 7 adalah contoh operator penugasan yang memberi nilai 7 di kanan ke variabel a yang ada di kiri.

d. Operator Logika: Operator yang digunakan untuk melakukan operasi logika

e. Operator Bitwise: Operator yang melakukan operasi bit terhadap operand. Operator ini beroperasi bit per bit sesuai dengan namanya.

f. Operator Identitas: Operator yang memeriksa apakah dua buah nilai atau variabel berada pada lokasi memori yang sama.

g. Operator Keanggotaan: Operator yang digunakan untuk memeriksa nilai atau variabel yang merupakan anggota dalam suatu data (string, list, tuple, set, dan dictionary).

```
19    X = 'PenjualanHarian'
20    Y = 'PenjualanBulanan'
21    Z = 'rugi'
22
23    print("X is Y:", X in Y)
24    print("X not in Z:", X not in Z)
X is Y: False
X not in Z: True
```

2.4 Tipe Data Bentukan

Terdapat empat tipe dengan penggunaan yang berbeda-beda, yakni list, tuple, set, dan dictionary. List merupakan sebuah kumpulan data yang terurut, dapat diubah, dan memungkinkan ada anggota yang sama. Tuple merupakan kumpulan data yang terurut, tidak dapat diubah, dan memungkinkan ada anggota yang sama. Set merupakan kumpulan data yang tidak berurutan, tidak terindeks, dan tidak memungkinkan ada anggota yang sama. Dan dictionary merupakan kumpulan data yang tidak berurutan, dapat diubah, tidak memungkinkan ada anggota yang sam

2.5 Percabangan

a. Percabangan IF: digunakan dua variabel, a dan b, sebagai bagian dari pernyataan if untuk menguji apakah b lebih kecil dari a.

```
a = 20
b = 18
if b<a:
    print("b lebih kecil dari a")
```

b. Percabangan IF-ELSE: Untuk melanjutkan percabangan, digunakan IF-ELSE.

```
28    a = 2
29    b = 330
30    print("A") if a > b else print("B")
31
```

c. Percabangan IF-ELSE-IF: Untuk melanjutkan penyeleksian dari percabangan IF-ELSE.

```
32    a = 330
33    b = 330
34    print("A") if a > b else print("=") if a == b else print("B")
```

d. Nested IF: sebuah pernyataan if yang dapat dimiliki dalam pernyataan if

```
32    x = 41
33    if x > 10:
34     print("Nilai di atas 10,")
35     if x > 20:
36     print("Dan juga di atas 20!")
37     else:
38     print("Tapi tidak di atas 20.")
39
```

2.6 Perulangan

a. Perulangan For: Digunakan untuk iterasi pada urutan berupa list, tuple, atau string

```
for x in range(6):
 print(x)
```

b. Perulangan While: Dapat dilakukan perulangan selama kondisi tertentu terpenuhi

```
i = 1
while i < 6:
 print(i)
 i += 1
```

2.7 Fungsi

Fungsi digunakan untuk mengeksekusi suatu blok kode tanpa harus menulisnya berulang.

```
def my_function():
 print("Halo Python dari Fungsi")

my_function()
```

Modul 2

1. Kelas

sebuah blueprint (cetakan) dari objek (atau instance) yang ingin dibuat. Kelas berisi dan mendefinisikan atribut/properti dan metode untuk objeknya nanti. Untuk membuat kelas, digunakan kata kunci class yang diikuti oleh nama kelas tersebut dan tanda titik dua.

```
class Manusia:
    ciri ='berat, tinggi'

    def _int_(ciri):
        print('ciri')
```

a. Atribut/Property: Variabel yang berada di dalam kelas yang terdiri atas dua jenis, yakni atribut kelas dan atribut objek.

```
class Kucing:
  warna = None
  usia = None
kucing1 = Kucing()
kucing1_warna = "hitam"
kucing2 = Kucing()
kucing2_warna = "putih"
print(kucing1_warna, kucing2_warna)
```

b. Method: Suatu fungsi di dalam kelas yang dapat diibaratkan sebagai sebuah proses yang dapat dilakukan oleh sebuah objek

```
class Kucing:
  warna = None
  usia = None
kucing1 = Kucing()
kucing1_warna = "hitam"
kucing2 = Kucing()
kucing1.berjalan_melompat()
kucing2.rebahan()
print=kucing1.berjalan_melompat()
print=kucing2.rebahan()
```

2. Objek

Objek berfungsi sebagai pengganti pemanggilan sebuah kelas sehingga hanya dapat mewakili sebuah kelas saja. Untuk membuat sebuah objek yang mewakili kelas, dapat dilakukan dengan memanggil nama kelas yang diinginkan ditambah tanda kurung ().

```
18    nama_object = nama_Class()
19
```

3. Magic Method

Metode yang diawali dan diakhiri dengan double underscore (dunder). Tujuan penggunaan magic method adalah untuk mengubah sifat bawaan suatu objek.

```
20    def __init__(self,bumi):
21        self.bumi=alam
22    def __add__(self,objek):
23        return self.bumi + objek.bumi
24
```

4. Konstruktor

Konstruktor adalah method yang pasti dijalankan secara otomatis pada saat sebuah objek dibuat untuk mewakili kelas tersebut. Selain operasi method dasar, konstruktor dapat menerima argumen yang diberikan ketika objek dibuat.

```
18    class HaloDunia:
19
20      def __init__(self):
21        print('Halo dunia')
```

5. Destruktor

Destruktor adalah fungsi yang dipanggil ketika user menghapus objek yang bertujuan untuk melakukan final cleaning up terakhir sebelum sebuah objek dihapus permanen.

```
20    def __init__(self,bumi):
21        self.bumi=alam
22    def __add__(self,objek):
23        print("objek{self.bumi}dihapus")
24
```

6. Setter dan Getter

Setter dan getter digunakan untuk melakukan enkapsulasi agar tidak terjadi perubahan data secara tidak sengaja. Setter adalah metode untuk menetapkan nilai atribut khususnya atribut private dan protected, sedangkan getter digunakan untuk mengambil nilai.

```
25    class Label:
26        def __init__(self, text, font):
27            self._text = text
28            self._font = font
29
30        def get_text(self):
31            return self._text
32
33        def set_text(self, value):
34            self._text = value
35
36        def get_font(self):
37            return self._font
38
39        def set_font(self, value):
40            self._font = value
```

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).

```
42    def make_pretty(func):
43        def inner():
44            print("I got decorated")
45            func()
46        return inner
47
48
49    def ordinary():
50        print("I am ordinary")
51
```

Modul 3

1. Abstraksi

Abstraksi merupakan sebuah konsep OOP dimana model yang dibuat hanya memperlihatkan atribut yang esensial dan menyembunyikan detail-detail yang tidak penting dari user.

2. Enkapsulasi (Encapsulation)

Enkapsulasi adalah sebuah metode yang digunakan untuk mengatur struktur kelas dengan cara menyembunyikan alur kerja dari kelas tersebut.

a. Public Access Modifier: Pendeklarasian suatu variabel atau method, dimana setiap class, variable dan method yang dibuat secara default merupakan public.

```
 8    class Geek:
 9        def __init__(self, name, age):
10            self.geekName = name
11            self.geekAge = age
12        def displayAge(self):
13            print("Age: ", self.geekAge)
14
15    obj = Geek("R2J", 20)
16    print("Name: ", obj.geekName)
17
```

b. Protected Access Modifier: Jika suatu variabel dan method dideklarasikan secara protected, maka Ia hanya dapat diakses oleh kelas turunannya. Caranya adalah dengan menambahkan underscore sebelum variable atau method.

```
18    class Student:
19        _name = None
20        _roll = None
21        _branch = None
22
23        def __init__(self, name, roll, branch):
24            self._name = name
25            self._roll = roll
26            self._branch = branch
27            print("Roll: ", self._roll)
28            print("Branch: ", self._branch)
```

c. Private Access Modifier: Variable dan method hanya dapat diakses dalam kelas itu sendiri sehingga menjadi lebih aman dengan menambahkan double underscore (__) sebelum nama variable dan methodnya.

```
class Geek:
      __name = None
      __roll = None
      def __init__(self, name, roll):
            self.__name = name
            self.__roll = roll
      def __displayDetails(self):
            print("Name: ", self.__name)

```

d. Setter dan Getter Tanpa Decorator: Digunakan untuk melakukan enkapsulasi

```
class manusia:
    def __init__(self, normh=64):
        self.normh=normh
#getter
def get_normh(self):
    return self._normh
#setter
def set_normh(self, x):
    self._normh=x
raj=manusia()
raj.set_normh(64)
print(raj.get_normh())
print(raj._normh)

```

e. Setter dan Getter dengan Decorator: Penggunaan setter getter dengan decorator hanya dapat dilakukan terhadap properti dengan access modifier protected dan private.

```
class manusia:
    def __init__(self, normh=64):
        self.__normh=normh
#getter
def get_normh(self):
    print("fungsi getter normh bayu")
    return self._normh
#setter
def set_normh(self, x):
    print("fungsi getter normh bayu")
    self._normh=x
bayu=manusia("bayu")
print(bayu.normh)
```

3. Object

a. Membuat Instance Objects: Bentuk umum dari instansiasi objek yaitu:

```
#nama_objek=Mahasiswa(name, nim)

mahasiswa_A = Mahasiswa("kiki", 20)

```

b. Mengakses Atribut Objek: Dapat dilakukan pengaksesan atribut dari objek dengan menggunakkan operator dot(titik)

```
mahasiswa1.nimdanalamat()
mahasiswa2.nimdanalamat()
print("data diti mahasiswa")

```

c. Menambah, menghapus, dan Mengubah atribut object: Objek yang sudah dibuat dapat dimodifikasi seperti ditambah, dihapus, ataupun diubah atributnya.

```
60  Keluarga = jumlahanak("Rahmat",3)
61  print("sebelum diubah")
62  Keluarga.jumlahanak = 7
63  print("setelah diubah")
64  Keluarga.jumlahanak()
65
```

Modul 4

1. Inheritance (Pewarisan): Pada inheritance, dapat dilakukan penurunan kelas dari kelas lain untuk hirarki kelas yang saling berbagi atribut dan metode

```
66  class Orang:
67
68    def __init__ (self, nama, asal):
69      self.nama = nama
70      self.asal = asal
71
72    def perkenalan (self):
73      print(f'Perkenalkan nama saya {self.nama} dari {self.asal}')
74
```

2. Polymorphism: Kemampuan method untuk bekerja dengan lebih dari satu tipe argumen.

```
75  class Tomato():
76      def type(self):
77        print("Vegetable")
78      def color(self):
79        print("Red")
80  class Apple():
81      def type(self):
82        print("Fruit")
83      def color(self):
84        print("Red")
85
86  def func(obj):
87        obj.type()
88        obj.color()
89
90  obj_tomato = Tomato()
91  obj_apple = Apple()
92  func(obj_tomato)
93  func(obj_apple)
```

3. Override/Overriding: Konsep ini dapat menimpa suatu metode yang ada pada parent class dengan mendefinisikan kembali method dengan nama yang sama pada child class.

```
95  class bangunruang():
96      def tampil(self):
97          print("bangunruangkubus")
98  class kerucut(bangunruang):
99      def tampil(self):
100         print("ini kerucut")
101 P1 = kerucut()
102 P1.tampil()
```

4. Overloading: Metode overloading mengizinkan sebuah class untuk memiliki fungsi dengan nama yang sama dan argumen yang berbeda. Secara umum overloading memiliki beberapa signature, yaitu jumlah argumen, tipe argumen, tipe keluaran dan urutan argumen.

```
104     class sumClass:
105         def sum(self, a, b):
106             print("First method:",a+b)
107         def sum(self, a, b, c):
108             print("Second method:", a + b + c)
109
110     obj=sumClass()
111     obj.sum(19, 8, 77) #correct output
112     obj.sum(18, 20) #throws error
```

5. Multiple Inheritance: Sebuah kelas yang dapat diturunkan lebih dari satu superclass.

```
114     class Mammal:
115         def mammal_info(self):
116             print("Mammals can give direct birth.")
117     class WingedAnimal:
118         def winged_animal_info(self):
119             print("Winged animals can flap.")
120     class Bat(Mammal, WingedAnimal):
121         pass
122     # create an object of Bat class
123     b1 = Bat()
124     b1.mammal_info()
125     b1.winged_animal_info()
```

6. Method Resolution Order di Python: Urutan pencarian metode dalam hirarki class yang berguna dalam multiple inheritance. Urutan MRO dalam python yaitu bawah-atas dan kiri-kanan.

```
127     class A:
128         def myname(self):
129             print("I am a class A")
130     class B(A):
131         def myname(self):
132             print("I am a class B")
133     class C(A):
134         def myname(self):
135             print("I am a class C")
136     c = C()
137     print(c.myname())
138
```

7. Dynamic Cast: Proses untuk mengubah nilai dari satu tipe data ke tipe data lainnya seperti dari string ke int atau sebaliknya. Ada 2 tipe konversi yaitu:

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

```
144     # Python automatically converts
145     # a to int
146     a = 7
147     print(type(a))
148
149     # Python automatically converts
150     # b to float
151     b = 3.0
152     print(type(b))
153
154     # Python automatically converts
155     # c to float as it is a float addition
156     c = a + b
157     print(c)
158     print(type(c))
159
160     # Python automatically converts
161     # d to float as it is a float multiplication
162     d = a * b
163     print(d)
164     print(type(d))
165
```

b. 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.

```
166     # Python program to demonstrate
167     # type Casting
168
169     # int variable
170     a = 5
171
172     # typecast to float
173     n = float(a)
174
175     print(n)
176     print(type(n))
177
```

8. Casting

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

```
144     class hewan:
145         def __init__(self, buas, jinak):
146             self.buas = buas
147             self.jinak = jinak
148         def jenisa(self):
149             print(f"{self.buas}{self.jinak}({self.tempathidup})")
150     class tempathidup(hewan):
151         def __init__(self, buas, jinak, tempathidup):
152             super().__init__(buas, jinak)
153             self.tempathidup = tempathidup()
154     Singa = tempathidup("singa","carnivora","hutan")
155     Singa.deskripsi
```

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

```
144     class hewan:
145         jinak = "Kucing"
146
147         def __init__(self, buas, jinak):
148             self.buas = buas
149             self.jinak = jinak
150         def jenisa(self):
151             print(f"{self.buas}{self.jinak}({self.tempathidup})")
152     class tempathidup(hewan):
153         def __init__(self, buas, jinak, tempathidup):
154             super().__init__(buas, jinak)
155             self.tempathidup = tempathidup()
156         def __deskripsi(self):
157             print(f"{self.buas}{super().jinak}({self.tempathidup})")
158     Singa = tempathidup("singa","carnivora","hutan")
159     Singa.deskripsi
```

c. Type casting: Konversi tipe kelas agar memiliki sifat/perilaku tertentu yang secara default tidak dimiliki kelas tersebut.

```
144     class hewan:
145
146         def __init__(self, buas, jinak):
147             self.buas = buas
148             self.jinak = jinak
149         def __str__(self):
150             return f"{self.__buas}({self.__jinak}) ({self.tempathidup})"
151
152         def __init__(self, buas, jinak, tempathidup):
153             return self.__jinak
154
155     Singa = tempathidup("singa","carnivora","hutan")
156     Singa.deskripsi
```