RESUME PRAKTIKUM MODUL 1-4

PEMROGRAMAN BERORIENTASI OBJEK

Disusun Oleh:

Nama : Abira Husnia

NIM : 121140113

Kelas : RB

PROGRAM STUDI TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI SUMATERA

LAMPUNG SELATAN

2023

DAFTAR ISI

PENGENALAN DAN DASAR PEMROGRAMAN PYTHON I

1. Pengenalan Bahasa Pemrograman Python

Python adalah bahasa pemrograman tingkat tinggi yang populer dan serbaguna. Dibuat pada tahun 1989 oleh Guido van Rossum, Python didesain untuk mudah dibaca, ditulis, dan dipahami. Python populer di kalangan pengembang web, ilmu data, dan pemrosesan bahasa alami. Ia memiliki sintaks yang sederhana dan jelas, sehingga mudah dipelajari bahkan bagi pemula. Beberapa kelebihan Python meliputi kemampuan untuk memproses data dan melakukan analisis numerik dengan mudah, memiliki banyak perpustakaan dan kerangka kerja (framework) yang berguna, serta dapat digunakan untuk membuat berbagai jenis aplikasi seperti aplikasi desktop, web, maupun mobile. Python dapat dijalankan pada berbagai sistem operasi seperti Windows, macOS, dan Linux. Python juga bersifat open source dan gratis untuk digunakan dan didistribusikan, bahkan untuk keperluan komersial.

2. Dasar Pemrograman Python

2.1 Variabel dan Tipe Data Primitif

Variabel digunakan untuk menyimpan nilai atau objek dalam program. Python memiliki aturan penamaan variabel yang cukup fleksibel, di mana nama variabel harus diawali dengan huruf atau garis bawah (_), dan dapat terdiri dari huruf, angka, dan garis bawah. Python memiliki beberapa tipe data dasar, seperti string, integer, float, boolean, dan lain-lain. Setiap tipe data memiliki cara penulisan dan penggunaan yang berbeda.

2.2 Operator

Python memiliki beberapa operator, yaitu operator aritmatika, operator perbandingan, operator penugasan, operator logika, operator bitwise, operator identitas, dan operator keanggotaan. Contoh:

- Operator aritmatika

```python
x = 11
y = 7

# Penjumlahan
print(x + y) # Output: 18

# Pengurangan
print(x - y) # Output: 4

# Perkalian
print(x * y) # Output: 77

# Pembagian
print(x / y) # Output: 1.5714285714285714

# Modulus (Sisa pembagian)
print(x % y) # Output: 4

# Pangkat
print(x ** y) # Output: 19487171
```

- Operator perbandingan

```python
x = 11
y = 7

# Lebih besar dari
print(x > y) # Output: True

# Lebih kecil dari
print(x < y) # Output: False

# Sama dengan
print(x == y) # Output: False

# Tidak sama dengan
print(x != y) # Output: True

# Lebih besar dari atau sama dengan
print(x >= y) # Output: True

# Lebih kecil dari atau sama dengan
print(x <= y) # Output: False
```

- Operator penugasan

```python
x = 11
y = 7

# Penugasan
z = x + y
print(z) # Output: 18

# Penugasan gabungan
x += y
print(x) # Output: 18

# Penugasan gabungan
x *= y
print(x) # Output: 126

# Penugasan gabungan
x %= y
print(x) # Output: 0
```

- Operator logika

```python
x = 11
y = 7

# AND
print(x > 5 and y < 10) # Output: True

# OR
print(x > 5 or y > 10) # Output: True

# NOT
print(not(x > 5 and y < 10)) # Output: False
```

- Operator bitwise

```python
x = 11
y = 7

# AND bitwise
print(x & y) # Output: 3

# OR bitwise
print(x | y) # Output: 15

# XOR bitwise
print(x ^ y) # Output: 12

# Shift left
print(x << y) # Output: 1408

# Shift right
print(x >> y) # Output: 0
```

- Operator identitas

```python
x = [1, 2, 3]
y = [1, 2, 3]
z = x

print(x is y)     # False
print(x is z)     # True
print(x is not y) # True
```

- Operator keanggotaan

```python
x = [1, 2, 3, 4, 5]

print(3 in x)    # True
print(7 in x)    # False
print(5 not in x) # False
```

2.3 Tipe Data Bentukan

Tipe data bentukan atau data struktur dalam Python adalah jenis variabel yang dapat menyimpan sejumlah nilai dan diakses dengan cara tertentu. Berikut adalah beberapa tipe data bentukan dalam Python:

- List: List adalah tipe data yang memungkinkan Anda untuk menyimpan banyak nilai dalam satu variabel. Anda dapat mengakses nilai dalam list dengan menggunakan indeks. List dideklarasikan dengan tanda kurung siku ([]).
- Tuple: Tuple adalah tipe data yang mirip dengan list, tetapi isinya tidak dapat diubah setelah dideklarasikan. Tuple dideklarasikan dengan tanda kurung biasa ().
- Set: Set adalah tipe data yang memungkinkan Anda untuk menyimpan sekumpulan nilai yang unik. Set dideklarasikan dengan menggunakan kurung kurawal ({ }).

- Dictionary: Dictionary adalah tipe data yang memungkinkan Anda untuk menyimpan kunci dan nilai dalam satu variabel. Kunci digunakan untuk mengakses nilai dalam dictionary. Dictionary dideklarasikan dengan tanda kurung kurawal ({}) dan kunci dan nilai dipisahkan dengan tanda titik dua (:).

2.4 Percabangan

Percabangan digunakan untuk mengatur alur program berdasarkan kondisi tertentu. Dalam Python, percabangan ditulis dengan menggunakan kata kunci if, elif, dan else.

- Percabangan if

```
x = 7

if x > 0:
    print("x adalah bilangan positif")
```

- Percabangan if-else

```
x = 7

if x > 0:
    print("x adalah bilangan positif")
else:
    print("x bukan bilangan positif")
```

- Percabangan if-else-if

```
x = 7

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

- Nested if

```
nilai = 75

if nilai >= 60:
    print("Anda lulus ujian")
    if nilai >= 80:
        print("Dan mendapat nilai A")
    elif nilai >= 70:
        print("Dan mendapat nilai B")
    else:
        print("Dan mendapat nilai C")
else:
    print("Anda tidak lulus ujian")
```

2.5 Perulangan

Perulangan digunakan untuk mengulang suatu blok kode program secara berulang-ulang. Dalam Python, perulangan dapat ditulis dengan menggunakan kata kunci for dan while.

- Perulangan for

```
buah = ['apel', 'jeruk', 'mangga', 'pisang']
for b in buah:
    print(b)
```

- Perulangan while

```
x = 0
while x < 5:
    print("Nilai x adalah", x)
    x += 1
```

2.6 Fungsi

Fungsi digunakan untuk membagi program menjadi bagian-bagian yang lebih kecil dan lebih terorganisir. Dalam Python, fungsi ditulis dengan menggunakan kata kunci def. Contoh:

```
def hitung_luas_persegi_panjang(panjang, lebar):
  luas = panjang * lebar
  return luas

hasil = hitung_luas_persegi_panjang(7, 3)
print(hasil)    # hasilnya 21
```

OBJEK DAN KELAS DALAM PYTHON (KONSTRUKTOR, SETTER, DAN GETTER)

1. Kelas

Kelas (class) dalam bahasa pemrograman Python adalah sebuah blueprint atau cetak biru yang mendefinisikan atribut dan metode (fungsi) dari sebuah objek. Dengan kata lain, kelas adalah sebuah struktur data yang dapat digunakan untuk membuat objek. Objek adalah instance (turunan) dari sebuah kelas, yang memiliki atribut dan metode yang ditentukan oleh kelas. Contoh:

```
1  class Mobil:
2    def __init__(self, merk, warna, tahun):
3      self.merk = merk
4      self.warna = warna
5      self.tahun = tahun
6
7    def jalankan(self):
8      print("Mobil", self.merk, "dijalankan")
```

Di dalam kelas pada Python, variabel atau atribut adalah objek yang terikat dengan objek kelas tersebut, dan memungkinkan untuk menyimpan data yang berkaitan dengan objek tersebut. Setiap objek kelas dapat memiliki atribut yang berbeda-beda, tergantung pada apa yang diperlukan untuk merepresentasikan objek tersebut. Atribut dapat diakses dan dimanipulasi melalui objek kelas tersebut. Pada contoh di atas, **merk**, **warna**, dan **tahun** adalah atribut dari kelas **Mobil**.

Fungsi atau method pada kelas adalah blok kode yang terkait dengan kelas tersebut dan dapat dipanggil untuk melakukan tindakan atau operasi tertentu pada objek kelas. Fungsi pada kelas juga dapat menerima input atau argumen, seperti halnya fungsi pada umumnya. Pada contoh di atas, **jalankan()** adalah sebuah method pada kelas **Mobil** yang dapat dipanggil untuk menjalankan mobil dan akan mencetak pesan "Mobil {merk} dijalankan", dimana **{merk}** adalah nilai dari atribut **merk** pada objek kelas tersebut.

2. Objek

Pada dasarnya, semua objek dalam Python adalah turunan dari kelas dasar Python yang disebut "object". Objek adalah sesuatu yang “mewakili” kelas. Objek berfungsi sebagai pengganti pemanggilan sebuah kelas, sebuah objek hanya dapat mewakili sebuah kelas saja. Contoh:

```
10 balok = Balok(5, 6, 7)
```

3. Magic Method

Magic method adalah metode khusus yang dimulai dengan double underscore (dunder) atau "__". Metode ini juga dikenal sebagai metode khusus, metode ajaib, atau metode operator overload. Metode ini digunakan untuk memberikan perilaku khusus pada objek Python, seperti operasi matematika, pemanggilan fungsi, pengindeksan, penggunaan operator perbandingan, dan lain-lain. Contoh:

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

    def __str__(self):
        return f"Name: {self.nama}, Kelas: {self.kelas}"

data1 = Data("Abira Husnia", "RB")
print(data1) # Output: Nama: Abira Husnia, Kelas : RB
```

4. Konstruktor

Konstruktor adalah metode khusus yang digunakan untuk menginisialisasi objek yang dibuat dari suatu kelas. Konstruktor biasanya ditulis **__init__()** dan dijalankan secara otomatis saat objek baru dibuat. Konstruktor digunakan untuk memberikan nilai awal pada atribut objek dan melakukan tugas-tugas lain yang diperlukan saat objek dibuat. Setiap kali objek dibuat dari kelas yang sama, konstruktor akan dipanggil untuk menginisialisasi objek tersebut. Contoh:

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

data1 = Data("Abira", "RB")
data2 = Data("Husnia", "RA")

print(data1.nama)  # Output: Abira
print(data1.kelas) # Output: RB
print(data2.nama)  # Output: Husnia
print(data2.kelas) # Output: RA
```

Dalam contoh di atas, kita membuat kelas **Data** dengan konstruktor yang menerima dua parameter yaitu **nama** dan **kelas**. Saat objek **data1** dan **data2** dibuat, konstruktor **__init__()** dipanggil secara otomatis untuk menginisialisasi atribut **nama** dan **kelas** pada objek tersebut.

5. Destruktor

Destruktor adalah metode khusus dalam Python yang digunakan untuk membersihkan sumber daya yang digunakan oleh objek ketika objek tersebut dihapus dari memori. Destruktor ditulis dengan **__del__()**. Destruktor tidak selalu dibutuhkan pada setiap kelas di Python, namun diperlukan jika kelas tersebut menggunakan sumber daya seperti koneksi database atau membuka file yang harus dibersihkan ketika objek dihapus.

```python
class Data:
    def __init__(self, nama):
        self.nama = nama
        print(f"{self.nama} has been created.")

    def __del__(self):
        print(f"{self.nama} has been deleted.")

data1 = Data("Abira")
data2 = Data("Husnia")

del data1
```

Dalam contoh di atas, kita membuat kelas **Data** dengan konstruktor yang menerima satu parameter yaitu **nama**. Ketika objek **data1** dan **data2** dibuat, konstruktor **__init__()** dipanggil secara otomatis untuk menginisialisasi atribut **nama** pada objek tersebut. Kemudian, ketika objek **data1** dihapus dengan perintah **del**, destruktor **__del__()** dipanggil secara otomatis untuk membersihkan sumber daya yang digunakan oleh objek tersebut. Dalam contoh ini, destruktor hanya mencetak pesan ke konsol bahwa objek telah dihapus.

6. Setter dan Getter

Setter dan getter adalah metode yang digunakan untuk mengakses dan memodifikasi atribut pada objek kelas. Setter digunakan untuk mengubah nilai dari atribut, sedangkan getter digunakan untuk mengambil nilai dari atribut. Dalam Python, setter dan getter biasanya diimplementasikan sebagai properti. Contoh:

```python
class Data:
    def __init__(self, nama, nim):
        self.__nama = nama
        self.__nim = nim

    def set_nama(self, nama):
        self.__nama = nama

    def set_nim(self, nim):
        self.__nim = nim

    def get_nama(self):
        return self.__nama

    def get_nim(self):
        return self.__nim

data = Data("Abira Husnia", "121140113")
print("Nama :", data.get_nama())
print("NIM :", data.get_nim())

data.set_nama("Abira Husnia")
data.set_nim("121140113")
print("Nama :", data.get_nama())
print("NIM :", data.get_nim())
```

Dalam contoh di atas, terdapat kelas **Data** yang memiliki properti **nama** dan **nim**. Properti-properti tersebut diakses melalui setter dan getter, yaitu **set_nama**, **set_nim**, **get_nama**, dan **get_nim**. Properti **nama** dan **nim** dideklarasikan sebagai private dengan menambahkan dua garis bawah (__) sebelum nama properti tersebut.

Pada saat objek **data** diinisialisasi, nilai **nama** dan **nim** diatur ke **"Abira Husnia"** dan **"121140113"**, masing-masing. Kemudian, nilai **nama** dan **nim** diubah dengan memanggil setter yang sesuai, yaitu **set_nama** dan **set_nim**. Nilai properti tersebut kemudian dicetak dengan memanggil getter yang sesuai, yaitu **get_nama** dan **get_nim**.

7. Decorator

Decorator dalam Python adalah fitur yang memungkinkan kita untuk memodifikasi atau menambahkan fungsi pada fungsi atau kelas yang sudah ada. Decorator dapat digunakan untuk mengubah perilaku fungsi atau kelas tanpa mengubah kode sumber asli dari fungsi atau kelas tersebut. Decorator ditandai dengan penggunaan karakter "@" diikuti dengan nama decorator. Decorator dapat didefinisikan sebagai fungsi yang mengambil sebuah fungsi sebagai argumen, dan mengembalikan fungsi yang baru dengan modifikasi atau tambahan fungsi. Contoh:

```python
def my_decorator(func):
    def wrapper():
        print("Sebelum fungsi dieksekusi.")
        func()
        print("Setelah fungsi dieksekusi.")
    return wrapper

@my_decorator
def say_hello():
    print("Hello, World!")

say_hello()
```

Pada contoh di atas, kita membuat decorator sederhana dengan nama **my_decorator**. Decorator tersebut menerima sebuah fungsi sebagai argumen dan mengembalikan sebuah fungsi baru yang menjadi wrapper dari fungsi yang didekorasi. Kemudian mendekorasi fungsi **say_hello** dengan menggunakan decorator **my_decorator** dengan menambahkan tanda **@** sebelum nama decorator pada baris yang sama dengan definisi fungsi **say_hello**. Fungsi **say_hello** akan dimodifikasi oleh decorator sehingga saat dipanggil, akan mencetak teks "Sebelum fungsi dieksekusi.", menjalankan fungsi yang asli, dan mencetak teks "Setelah fungsi dieksekusi". Ketika fungsi **say_hello** dipanggil pada baris terakhir, maka fungsi tersebut akan secara otomatis dijalankan melalui wrapper yang sudah dimodifikasi oleh decorator.

ABSTRAKSI DAN ENKAPSULASI (VISIBILITAS FUNGSI DAN VARIABEL, RELASI ANTAR KELAS)

1. Abstraksi

Abstraksi adalah konsep pemrograman yang memungkinkan kita untuk menyembunyikan detail atau kompleksitas yang tidak perlu dari sebuah objek atau program, dan hanya menampilkan fungsionalitas yang penting atau relevan untuk pengguna. Dalam Python, abstraksi biasanya dicapai melalui penggunaan kelas dan metode. Kita dapat mendefinisikan kelas dengan metode yang hanya menampilkan fungsionalitas yang penting dan menyembunyikan detail yang kompleks. Dengan cara ini, pengguna hanya perlu mengetahui cara menggunakan kelas dan metode tersebut tanpa perlu memahami implementasi yang kompleks di baliknya. Contoh:

```python
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

class Square(Shape):
    def __init__(self, side):
        self.side = side

    def area(self):
        return self.side * self.side

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius * self.radius

square = Square(3)
circle = Circle(7)

print("Luas Persegi:", square.area())
print("Luas Lingkaran:", circle.area())
```

2. Enkapsulasi (Encapsulation)

Enkapsulasi adalah konsep pemrograman yang memungkinkan kita untuk menyembunyikan detail implementasi dari sebuah kelas atau objek, dan hanya mengekspos fungsionalitas yang relevan untuk pengguna. Berikut adalah 3 jenis access modifier yang terdapat dalam Python beserta contohnya:

- Public

Tidak ada tanda underscore yang digunakan untuk menandakan aksesibilitas dari atribut atau method. Semua elemen kelas dapat diakses dari mana saja, baik dari dalam kelas maupun luar kelas. Contoh:

```python
class Data:
    def __init__(self, nama, kelas):
        self.nama = nama # public attribute
        self.kelas = kelas # public attribute

    def say_hello(self):
        print("Hai", self.nama,"!")

data = Data("Abira", "RB")
print(data.nama) # mengakses public attribute
data.say_hello() # mengakses public method
```

- Protected

Atribut atau method diawali dengan satu tanda underscore (_) untuk menandakan bahwa aksesibilitasnya terbatas. Atribut atau method yang diawali dengan underscore dapat diakses dari dalam kelas itu sendiri dan juga kelas turunannya. Contoh:

```python
class Data:
    def __init__(self, nama):
        self._nama = nama # protected attribute

    def say_hello(self):
        print("Hai", self._nama,"!")

class Mahasiswa(Data):
    def __init__(self, nama, kelas):
        super().__init__(nama)
        self.kelas = kelas # public attribute

    def info(self):
        self.say_hello() # mengakses protected method
        print("Kamu seorang mahasiswa dan berada di kelas", self.kelas)

mhs = Mahasiswa("Abira", "RB")
mhs.info() # mengakses public method
```

- Private

Atribut atau method diawali dengan dua tanda underscore (__). Aksesibilitasnya sangat terbatas karena hanya bisa diakses di dalam kelas itu sendiri dan tidak bisa diakses oleh kelas turunannya maupun dari luar kelas. Namun, meskipun bersifat private, atribut atau method masih bisa diakses dengan menggunakan konsep name mangling. Contoh:

```python
class Data:
    def __init__(self, nama):
        self.__nama = nama # private attribute

    def say_hello(self):
        print("Hai", self.__nama,"!")

class Mahasiswa(Data):
    def __init__(self, nama, kelas):
        super().__init__(nama)
        self.kelas = kelas # public attribute

    def info(self):
        self.say_hello() # mengakses private method
        print("Kamu seorang mahasiswa dan berada di kelas", self.kelas)

mhs = Mahasiswa("Abira", "RB")
mhs.info() # mengakses public method
```

PEWARISAN DAN POLIMORFISME (OVERLOADING, OVERRIDING, DYNAMIC CAST)

1. Inheritance (Pewarisan)

Inheritance atau pewarisan adalah konsep dalam pemrograman berorientasi objek dimana sebuah kelas dapat menurunkan sifat-sifat atau method-method yang dimilikinya kepada kelas turunannya. Dalam Python, kita bisa menerapkan inheritance dengan menggunakan keyword class diikuti dengan nama kelas yang diturunkan dan nama kelas induk yang diwarisi sifat-sifat atau method-methodnya. Contoh:

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

    def bersuara(self):
        print("Bersuara...")

class Kucing(Hewan):
    def __init__(self, nama):
        Hewan.__init__(self, "Kucing")
        self.nama = nama

    def bersuara(self):
        print(self.nama + " : Meow!")

kucing1 = Kucing("Catty")
kucing1.bersuara()
```

2. Polymorphism

Polymorphism adalah kemampuan sebuah objek untuk dapat memperlihatkan beberapa bentuk atau tampilan yang berbeda dari sebuah method atau atribut. Dalam Python, polimorfisme dapat diterapkan dengan menggunakan konsep pewarisan (inheritance) dan juga konsep abstract class. Contoh:

```python
class Kucing:
    def bersuara(self):
        print("Meow!")

class Anjing:
    def bersuara(self):
        print("Guk Guk!")

def main():
    hewan1 = Kucing()
    hewan2 = Anjing()

    hewan1.bersuara()
    hewan2.bersuara()

if __name__ == "__main__":
    main()
```

3. Override/Overriding

Override atau overriding adalah konsep dimana sebuah method di kelas turunan mengganti atau mengubah implementasi dari method yang sama yang sudah ada di kelas induknya. Dalam Python, override dapat dilakukan dengan membuat method yang sama dengan method di kelas induk dan menimpanya dengan implementasi yang berbeda. Contoh:

```python
class Kucing:
    def bersuara(self):
        print("Meow!")

class Anjing:
    def bersuara(self):
        print("Guk Guk!")

def main():
    hewan1 = Kucing()
    hewan2 = Anjing()

    hewan1.bersuara()
    hewan2.bersuara()

if __name__ == "__main__":
    main()
```

4. Overloading

Overloading adalah konsep dimana sebuah method memiliki nama yang sama dengan method lainnya namun memiliki parameter yang berbeda. Dalam Python, overloading tidak diterapkan secara langsung karena Python tidak mendukung adanya multiple method dengan nama yang sama. Namun, kita bisa melakukan overloading dengan menggunakan *args dan **kwargs. Contoh:

```python
class Calculator:
    def add(self, *args):
        result = 0
        for num in args:
            result += num
        return result

def main():
    calc = Calculator()
    print(calc.add(1, 2))
    print(calc.add(1, 2, 3))

if __name__ == "__main__":
    main()
```

5. Multiple Inheritance

Multiple inheritance adalah konsep dimana sebuah kelas dapat diturunkan dari lebih dari satu kelas induk. Dalam Python, kita bisa menerapkan multiple inheritance dengan menyertakan dua atau lebih kelas induk dalam deklarasi kelas turunan. Contoh:

```python
class Pekerja:
    def bekerja(self):
        print("Saya bekerja di kantor")

class Pelajar:
    def belajar(self):
        print("Saya belajar di sekolah")

class Penduduk(Pekerja, Pelajar):
    def beraktivitas(self):
        print("Saya adalah penduduk yang bisa bekerja dan belajar")

def main():
    penduduk = Penduduk()
    penduduk.bekerja()
    penduduk.belajar()
    penduduk.beraktivitas()

if __name__ == "__main__":
    main()
```

6. Method Resolution Order

Method Resolution Order (MRO) adalah urutan pencarian method yang dilakukan oleh interpreter ketika sebuah method dipanggil pada sebuah kelas dengan multiple inheritance. Dalam Python, MRO dihitung menggunakan algoritma C3 yang memastikan bahwa setiap method hanya dipanggil satu kali dan dalam urutan yang benar. Contoh:

```python
class A:
    def method(self):
        print("A.method()")

class B(A):
    def method(self):
        print("B.method()")

class C(A):
    def method(self):
        print("C.method()")

class D(B, C):
    pass

def main():
    obj = D()
    obj.method()

if __name__ == "__main__":
    main()
```

7. Dynamic Cast

Dynamic Cast adalah sebuah konsep dalam pemrograman dimana kita dapat mengubah tipe data suatu objek dari tipe yang satu ke tipe yang lain pada saat runtime. Konsep ini sering digunakan dalam pemrograman berorientasi objek dan biasanya melibatkan konversi dari tipe turunan ke tipe induk atau sebaliknya.

7.1 Implisit

Implisit adalah konversi yang dilakukan secara otomatis oleh Python saat kita melakukan operasi antara dua objek dengan tipe data yang berbeda. Python akan mengubah salah satu objek tersebut ke tipe data yang sama dengan objek lainnya agar operasi dapat dilakukan.

```
x = 5  # variabel x bertipe data integer
y = 2.5  # variabel y bertipe data float

hasil = x + y  # variabel hasil bertipe data float karena tipe data float memiliki preseden yang lebih tinggi daripada integer
print(hasil)  # output: 7.5
```

7.2 Eksplisit

Eksplisit adalah konversi yang dilakukan secara manual dengan mengubah tipe data suatu objek menggunakan fungsi bawaan Python. Explicit casting sangat berguna ketika kita ingin mengubah tipe data suatu objek agar dapat digunakan pada operasi atau fungsi tertentu.

```
x = "123"  # variabel x bertipe data string

int_x = int(x)  # variabel int_x bertipe data integer setelah dilakukan dynamic cast dari string ke integer
print(int_x)  # output: 123
```

8. Casting

Casting adalah sebuah proses konversi dari tipe data ke tipe data yang lain. Dalam Python, terdapat beberapa jenis casting, yaitu:

8.1 Downcasting

Upcasting adalah pengubahan tipe data objek ke tipe data yang lebih umum. Contohnya adalah mengubah tipe data dari anak kelas menjadi tipe data induk kelas. Contoh:

```
class Animal:
    def __init__(self, name):
        self.name = name

class Cat(Animal):
    def meow(self):
        print("Meow!")

cat = Cat("Catty")
animal = cat  # Upcasting
print(animal.name)  # Output: Catty
```

8.2 Upcasting

Downcasting adalah pengubahan tipe data objek ke tipe data yang lebih spesifik. Contohnya adalah mengubah tipe data dari induk kelas menjadi tipe data anak kelas. Contoh:

```python
class Animal:
    def __init__(self, name):
        self.name = name

class Cat(Animal):
    def meow(self):
        print("Meow!")

animal = Animal("Catty")
cat = Cat(animal.name)  # Downcasting
cat.meow()  # Output: Meow!
```

8.3 Type casting

Type casting adalah pengubahan tipe data objek dari satu tipe data ke tipe data lainnya. Contohnya adalah mengubah tipe data string menjadi integer atau float. Contoh:

```python
x = "7"
y = int(x)  # Type casting
print(y)  # Output: 7

a = 3.27
b = int(a)  # Type casting
print(b)  # Output: 3
```