TUGAS PRAKTIKUM
PEMROGRAMAN BERORIENTASI OBJEK

Disusun oleh:

Nama : AMDHAN ANGGORO

NIM : 121140226

Kelas : Pemrograman Berorientasi Objek RB

PROGRAM STUDI TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI SUMATERA

LAMPUNG SELATAN

2023

DAFTAR ISI

PEMBAHASAN

A. Konsep Abstraksi

Dalam konsep Object Oriented Program (OOP), abstraksi merupakan sesuatu yang dibuat hanya memperlihatkan atribut yang esensial dan menyembunyikan detail-detail yang tidak penting dari user. Dengan abstraksi kompleksitas akan berkurang. Dengan menerapkan abstraksi pengguna/user dapat mengetahui apa yang objek lakukan, tetapi tidak tau mekanisme yang sedang terjadi di belakang layar. Contohnya ketika mengendarai mobil, user mengetahui bagaimana menyalakan mobil, menjalankan, menghentikan, dan lainnya, tetapi tidak mengetahui mekanisme apa yang terjadi pada mobil ketika mendapat perintah tersebut.

1. Kelas Abstrak dalam Python

Kelas Abstrak merupakan sebuah class yang bersifat abstrak yang berarti kelas abstrak tidak bisa diinstansiasi (tidak bisa dibuat menjadi objek) langsung dan berperan sebagai 'kerangka dasar' bagi class turunannya. Kelas abstrak umumnya akan memiliki sebuah abstract method didalamnya. Abstract method merupakan sebuah method atau fungsi yang harus diimplementasikan ulang di dalam class anak (child class). Abstract method ditulis tanpa isi dari method, melainkan hanya 'signature'-nya saja. Signature dari sebuah method adalah bagian method yang terdiri dari nama method dan parameternya (jika ada).

Sub-kelas yang mewarisi kelas abstrak harus mengimplementasikan (OVERRIDE) semua fungsi abstrak yang didefinisikan kelas abstrak. Kelas Abstrak digunakan untuk membuat struktur logika penurunan di dalam pemrograman objek. Meskipun kelas abstrak tidak dapat dibuat objeknya secara langsung, dalam kelas abstrak dapat didefinisikan konstruktor yang dapat dipanggil dari sub-kelas yang mewarisi kelas abstrak untuk membuat objeknya. Kelas abstrak ditandai dengan mewarisi kelas ABC (Abstract Base Class). Contoh pemakaian kelas abstrak misalnya dapat dipakai pada konsep mobil, dimana tiap-tiap mobil tentunya memiliki suatu kesamaan namun implementasi yang berbeda (misal dari segi kecepatan, bahan bakar, dll).

2. Implementasi Kelas Abstrak dengan Modul ABC

Python memiliki modul untuk menggunakan ABC (Abstract Base Classes). Fungsi abstrak pada kelas abstrak ditandai dengan memberikan decorator @abstractmethod pada fungsi yang dibuat.

```python
from abc import ABC, abstractmethod

#Kelas Bentuk3D merupakan kelas abstrak dan tidak dapat diinstansiasi
class Bentuk3D(ABC):
    @abstractmethod
    def get_luas(self):
        #Fungsi ini wajib diimplementasikan pada child class
        pass

    @abstractmethod
    def get_volume(self):
        #Fungsi ini wajib diimplementasikan pada child class
        pass
```

```python
from abc import ABC, abstractmethod

#Kelas Bentuk3D merupakan kelas abstrak dan tidak dapat diinstansiasi
class Bentuk3D(ABC):
    @abstractmethod
    def get_luas(self):
        #Fungsi ini wajib diimplementasikan pada child class
        pass

    @abstractmethod
    def get_volume(self):
        #Fungsi ini wajib diimplementasikan pada child class
        pass

#Coba instasiasi kelas abstrak
bentuk1 = Bentuk3D()
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                    Python

PS C:\Users\User> & C:/Users/User/AppData/Local/Programs/Python/Python310/python.exe "d:/File Rani/
Prak PBO/Kelas Abstrak.py"
Traceback (most recent call last):
  File "d:\File Rani\Prak PBO\Kelas Abstrak.py", line 16, in <module>
    bentuk1 = Bentuk3D()
TypeError: Can't instantiate abstract class Bentuk3D with abstract methods get_luas, get_volume
```

Dari potongan kode tersebut akan menghasilkan error jika kelas Bentuk3D langsung dijadikan sebuah objek. Untuk mengatasi hal tersebut dan melakukan implementasi terhadap kelas abstrak diperlukan sebuah child class dari kelas abstrak tersebut.

Membuat child class untuk kelas abstrak sendiri ada ketentuannya, salah satunya yaitu ketika membuat child class dari kelas abstrak harus membuat kembali seluruh metode/fungsi yang ada pada parent class (kelas abstrak). Contohnya seperti berikut:

```python
class Balok(Bentuk3D):
    def __init__(self, panjang, lebar, tinggi):
        self.p = panjang
        self.l = lebar
        self.t = tinggi

    def get_luas(self):
        print(f"Luas   : {2 * ((self.p * self.l) + (self.p * self.t) + (self.l * self.t))}")

    def get_volume(self):
        print(f"Volume : {self.p * self.l * self.t}")

bentuk1 = Balok(4, 3, 5)
bentuk1.get_luas()
bentuk1.get_volume()
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                    Python

PS C:\Users\User> & C:/Users/User/AppData/Local/Programs/Python/Python310/python.exe "d:/File Rani/Prak PBO/Abstrak child
class.py"
Luas   : 94
Volume : 60
```

Selain fungsi seperti gambar tersebut, sebuah kelas abstrak juga dapat memiliki konstruktor dan fungsi biasa (yang tidak sepenuhnya kosong). Untuk membuat konstruktor dapat dilihat seperti berikut:

```python
from abc import ABC, abstractmethod

#Kelas Bentuk3D merupakan kelas abstrak dan tidak dapat diinstansiasi
class Bentuk3D(ABC):
    def __init__(self, warna):
        self.warna = warna

    @abstractmethod
    def get_luas(self):
        #Fungsi ini wajib diimplementasikan pada child class
        pass

    @abstractmethod
    def get_volume(self):
        #Fungsi ini wajib diimplementasikan pada child class
        pass

class Balok(Bentuk3D):
    def __init__(self, warna, panjang, lebar, tinggi):
        super().__init__(warna)
        self.p = panjang
        self.l = lebar
        self.t = tinggi

    def get_luas(self):
        print(f"Luas   : {2 * ((self.p * self.l) + (self.p * self.t) + (self.l * self.t))}")

    def get_volume(self):
        print(f"Volume : {self.p * self.l * self.t}")

bentuk1 = Balok("Biru", 4, 3, 5)
bentuk1.get_luas()
bentuk1.get_volume()
print(f"Warna  : {bentuk1.warna}")
```

```
PS C:\Users\User> & C:/Users/User/AppData/Local/Programs/Python/Python310/python.exe "d:/File Rani/Prak PBO/Abstrak child class.py"
Luas   : 94
Volume : 60
Warna  : Biru
```

Untuk implementasi fungsi biasa (konkret) dapat dilihat pada kelas abstrak dapat dilihat seperti contoh berikut (yaitu fungsi get_warna):

```python
from abc import ABC, abstractmethod

#Kelas Bentuk3D merupakan kelas abstrak dan tidak dapat diinstansiasi
class Bentuk3D(ABC):
    def __init__(self, warna):
        self.warna = warna

    @abstractmethod
    def get_luas(self):
        #Fungsi ini wajib diimplementasikan pada child class
        pass

    @abstractmethod
    def get_volume(self):
        #Fungsi ini wajib diimplementasikan pada child class
        pass

    def get_warna(self):
        print(f"Warna {self.warna}")

class Balok(Bentuk3D):
    def __init__(self, warna, panjang, lebar, tinggi):
        super().__init__(warna)
        self.p = panjang
        self.l = lebar
        self.t = tinggi

    def get_luas(self):
        print(f"Luas   : {2 * ((self.p * self.l) + (self.p * self.t) + (self.l * self.t))}")

    def get_volume(self):
        print(f"Volume : {self.p * self.l * self.t}")

bentuk1 = Balok("Biru", 4, 3, 5)
bentuk1.get_luas()
bentuk1.get_volume()
print(f"Warna  : {bentuk1.warna}")
bentuk1.get_warna()
```

```
PS C:\Users\User> & C:/Users/User/AppData/Local/Programs/Python/Python310/python.exe "d:/File Rani/Prak PBO/Abstrak child class.py"
Luas   : 94
Volume : 60
Warna  : Biru
Warna Biru
```

B. Interface

Interface adalah koleksi dari method atau fungsi-fungsi yang perlu disediakan oleh implementing class (child class). Interface mengandung metode/fungsi yang didefinisikan sebagai abstract. Metode abstract merupakan sebuah metode yang didefinisikan oleh interface dan interface dapat memberi arti yang konkret pada metode abstract interface. Metode abstract akan memiliki satu-satunya deklarasi karena tidak adanya implementasi. Pengimplementasian sebuah interface adalah sebuah cara untuk menulis kode yang elegan dan terorganisir.

Perbedaan Interface dengan Abstract Class pada Python

Interface	Abstract Class
Semua metode dari sebuah (formal) interface adalah abstrak	Sebuah abstract class dapat mempunyai metode abstract dan juga metode konkret

Digunakan jika semua fitur perlu diimplementasikan secara berbeda untuk objek yang berbeda	Digunakan saat ada beberapa fitur umum yang dimiliki oleh semua objek
Cenderung lebih lambat dibandingkan dengan abstract class	Lebih cepat (tidak semua implementasi perlu dituliskan pada child class)

1. **Informal Interface**

   Pengimplementasian interface di Python kadang lebih diartikan sebagai sebuah konsep tanpa aturan yang terlalu ketat. Informal interface merupakan kelas yang mendefinisikan metode atau fungsi-fungsi yang dapat di implementasi (override) tanpa adanya unsur paksaan (cukup diimplementasi jika dibutuhkan). Untuk mengimplementasikannya harus dibuat kelas konkret dulu. Kelas konkret merupakan subclass dari (abstrak) interface yang menyediakan implementasi dari method atau fungsi-fungsi di kelas interface. Untuk melakukan implementasi menggunakan inheritance.

   Contoh kasus informal interface adalah sebagai berikut:

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

    def __len__(self):
        return len(self.__list_hewan)

    def __contains__(self, hewan):
        return hewan in self.__list_hewan

class Kebun_Binatang(Hewan):
    pass

ragunan = Kebun_Binatang(["Gajah", "Singa", "Jerapah"])

#Cek Banyaknya Hewan
print(f"Banyak hewan : {len(ragunan)}")

#Cek Hewan Tertentu
print("Gajah" in ragunan)
print("Kucing" in ragunan)
print("Singa" not in ragunan)

#Iterasi List Hewan yang ada
for hewan in ragunan:
    print(hewan)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\Users\User> & C:/Users/User/AppData/Local/Programs/Python/Python310/python.exe "d:/File Rani/Prak PBO/Informal Interface.py"
Banyak hewan : 3
True
False
False
Traceback (most recent call last):
  File "d:\File Rani\Prak PBO\Informal Interface.py", line 25, in <module>
    for hewan in ragunan:
TypeError: 'Kebun_Binatang' object is not iterable
```

Dari program tersebut, kelas Kebun_Binatang dengan objek ragunan dapat mengecek banyaknya hewan (__len__) dan mengecek keberadaan hewan (__contains__) pada kebun binatang tersebut. Namun, ketika dibutuhkan list dari hewan-hewan (melalui iterasi) yang berada pada kelas Kebun_Binatang dengan objek ragunan, maka akan menghasilkan error.

Untuk mengatasi error tersebut, pada kelas Kebun_Binatang perlu diimplementasikan sebuah fungsi baru (__iter__) sehingga memungkinkan terjadinya iterasi untuk mendapatkan list hewan yang ada. Namun, implementasi fungsi __iter___ ini tidak dipaksakan oleh kelas parent dari Kebun_Binatang yaitu kelas Hewan. Karena hal tersebut konsep ini dikenal dengan informal interface. Hasilnya dapat dilihat seperti gambar berikut.

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

    def __len__(self):
        return len(self.__list_hewan)

    def __contains__(self, hewan):
        return hewan in self.__list_hewan

class Kebun_Binatang(Hewan):
    def __iter__(self):
        return iter(self._Hewan__list_hewan)

ragunan = Kebun_Binatang(["Gajah", "Singa", "Jerapah"])

#Cek Banyaknya Hewan
print(f"Banyak hewan : {len(ragunan)}")

#Cek Hewan Tertentu
print("Gajah" in ragunan)
print("Kucing" in ragunan)
print("Singa" not in ragunan)

#Iterasi List Hewan yang ada
for hewan in ragunan:
    print(hewan)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\Users\User> & C:/Users/User/AppData/Local/Programs/Python/Python310/
python.exe "d:/File Rani/Prak PBO/Informal Interface.py"
Banyak hewan : 3
True
False
False
Gajah
Singa
Jerapah
```

2. Formal Interface

Pada formal interface kelas parent (abstrak) dapat dibangun hanya dengan sedikit baris kode, untuk kemudian diimplementasikan pada kelas turunannya (konkret). Implementasi ini bersifat wajib/dipaksakan sehingga dinamakan formal interface. Salah satu cara membuat formal interface adalah dengan abstract class method seperti pada contoh kelas abstrak sebelumnya.

```python
from abc import ABC
from abc import abstractmethod

class FiturVoiceAssistance(ABC):
    @abstractmethod
    def aktifkan_asisten(self):
        pass
    @abstractmethod
    def matikan_asisten(self):
        pass
    @abstractmethod
    def greetings(self):
        pass

```

Contoh Kelas Abstrak

```python
class Smartphone(FiturVoiceAssistance):
    def aktifkan_asisten(self):
        status = "Asisten suara diaktifkan !"
        print(status)
    def matikan_asisten(self):
        status = "Asisten suara dimatikan !"
        print(status)
    def greetings(self):
        user = input("Masukan username : ")
        print("Halo ", user, " Selamat datang :)")

asus = Smartphone()
asus.aktifkan_asisten()
asus.greetings()
asus.matikan_asisten()
```

Contoh Kelas Konkret

```
PS C:\Users\User> & C:/Users/User/AppData/Local/Programs/Python/Python310/python.exe "d:/File
 Rani/Prak PBO/Formal Interface.py"
Asisten suara diaktifkan !
Masukan username : Khairani
Halo  Khairani  Selamat datang :)
Asisten suara dimatikan !
```

Hasil Output

C. Metaclass

Dalam bahasa pemrograman Python semua tipe pada dasarnya adalah suatu objek/kelas juga (termasuk int, float, dll).

```python
Angka = 500
phi = 3.14
Nama = "Khairani"

print("Tipe dari Angka : ", type(Angka))
print("Tipe dari phi   : ", type(phi))
print("Tipe dari Nama  : ", type(Nama))
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

PS C:\Users\User> & C:/Users/User/AppData/Local/Programs/Python/Python310
/python.exe "d:/File Rani/Prak PBO/Tipe data Metaclass.py"
Tipe dari Angka :  <class 'int'>
Tipe dari phi   :  <class 'float'>
Tipe dari Nama  :  <class 'str'>
```

Oleh karena itu, bisa saja dibuat sebuah kelas turunan (subclass) dari int atau tipe dasar lainnya yang memiliki karakteristik tambahan.

Konsep tipe ini bahkan juga berlaku pada objek dan kelas yang masih kosong seperti contoh

dibawah.

```
1   class Empty:
2       pass
3
4   orang = Empty()
5   print("Tipe dari orang : ", type(orang))
6   print("Tipe dari Empty : ", type(Empty))
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

PS C:\Users\User> & C:/Users/User/AppData/Local/Programs/Python/Python310
/python.exe "d:/File Rani/Prak PBO/Metaclass Empty.py"
Tipe dari orang :  <class '__main__.Empty'>
Tipe dari Empty :  <class 'type'>
```

Seperti yang terlihat pada output tersebut, tipe dari objek orang adalah "Empty" dan tipe dari kelas Empty adalah "type". Hal tersebu dapdat terjadi karena dalam Python, terdapat konsep hirarki dimana objek dibentuk dari kelas (class), sedangkan kelas dibentuk dari hirarki yang lebih tinggi yaitu metaclass (dalam hal ini berupa contoh metaclass tersebut adalah "type").

Dengan kata lain, metaclass adalah tipe kelas spesial yang berfungsi untuk membuat kelas- kelas pada Python (atau biasa disebut dengan *class factory*/pabrik kelas).

1. **Menggunakan type**

   Berikut merupakan sintaks membuat metaclass menggunakan type :

   ```
   type('NamaKelas', (Parent1,), {'nama_attr' : 'nilai_attr'})
   ```

   Dibawah ini merupakan contoh pembuatan metaclass menggunakan type. Jika variabel "keterangan" dipanggil, maka metaclass type tersebut akan membuat sebuah class bernama "Masalah", dengan tuple berisi nama parent class (dikosongkan saja jika class yang dibuat tidak melakukan inheritance dari class manapun), dan atribut dari kelas tersebut yang berbentuk dictionary, dengan program sebagai berikut:

   ```
   1  keterangan = type('Masalah', (), {'kejadian' : 'Kakak bertengkar dengan adik',
   2                                     'alasan' : 'Adik merusak boneka kesayangan kakak'})
   3
   4  print("Sebuah Kelas Telah Dibuat, Bernama : ", keterangan)
   5  print("Masalah apa yang sedang terjadi?   : ", keterangan.kejadian)
   6  print("Alasan kakak dan adik bertengkar   : ", keterangan.alasan)
   7  print()
   8  print(keterangan())
   ```

   ```
   PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

   PS C:\Users\User> & C:/Users/User/AppData/Local/Programs/Python/Python310
   /python.exe "d:/File Rani/Prak PBO/type Metaclass.py"
   Sebuah Kelas Telah Dibuat, Bernama :  <class '__main__.Masalah'>
   Masalah apa yang sedang terjadi?   :  Kakak bertengkar dengan adik
   Alasan kakak dan adik bertengkar   :  Adik merusak boneka kesayangan kakak

   <__main__.Masalah object at 0x00000215A35B7D60>
   ```

   Line terakhir merupakan alamat dari class "Masalah" yang telah dibuat. Dapat dilihat bahwa class "Masalah" merupakan sebuah "object" yang berada di alamat 0x00000215A35B7D60.

   Jika sebuah child class dibuat dan mewarisi parent class dapat dilihat dari program

lanjutan di bawah, jika “tambah_keterangan” dipanggil, maka metaclass akan membuat sebuah child class “Child_Masalah” yang mewarisi parent class “keterangan” pada contoh sebelumnya.

```
10  tambah_keterangan = type('Child_Masalah', (keterangan,),
11                           {'akibat' : 'Ibu memarahi keduanya dan menyuruh keduanya saling berbaikan'})
12
13  print("Sebuah Kelas Telah Dibuat, Bernama : ", tambah_keterangan)
14  print("Masalah apa yang sedang terjadi?   : ", tambah_keterangan.kejadian)
15  print("Alasan kakak dan adik bertengkar   : ", tambah_keterangan.alasan)
16  print("Akibat dari permasalahan tersebut  : ", tambah_keterangan.akibat)
17  print()
18  print(tambah_keterangan())
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

Sebuah Kelas Telah Dibuat, Bernama :  <class '__main__.Child_Masalah'>
Masalah apa yang sedang terjadi?   :  Kakak bertengkar dengan adik
Alasan kakak dan adik bertengkar   :  Adik merusak boneka kesayangan kakak
Akibat dari permasalahan tersebut  :  Ibu memarahi keduanya dan menyuruh keduanya saling berbaikan

<__main__.Child_Masalah object at 0x000001471D4EBFD0>
```

2. Menggunakan parameter metaclass

Selain langsung menggunakan type untuk membuat sebuah kelas, sebuah kelas turunan dapat juga dibuat hasil modifikasi terhadap type itu sendiri. Untuk memodifikasinya, perlu dibuat sebuah kelas turunan dengan basis dari type.

```
1   class Human(type):
2       def __new__(cls, nama_kelas, bases, attr_dict):
3           kelas = super().__new__(cls, nama_kelas, bases, attr_dict)
4           kelas.jenis_metaclass = "Human adalah Metaclass"
5           return kelas
6
7   class Person(metaclass = Human):
8       pass
9
10  orang = Person()
11  print(orang.jenis_metaclass)
```

Seperti yang diketahui, bahwa sebuah type adalah metaclass. Jika dibuat sebuah kelas turunan dari type, maka kelas turunan dari type ini juga merupakan sebuah metaclass. Dari gambar diatas, dibuat sebuah metaclass yang akan secara otomatis membuat atribut ‘jenis_metaclass’ dengan isi “Human juga Metaclass”

Secara default, metaclass yang digunakan saat membuat sebuah kelas adalah type. Setelah membuat sebuah metaclass hasil turunan dari type, maka kita dapat membuat sebuah kelas dengan metaclass hasil modifikasi sendiri. Output dari hasil program diatas sebagai berikut.

```
PS C:\Users\User> & C:/Users/User/AppData/Local/Programs/Python/Python310/
python.exe "d:/File Rani/Prak PBO/class Metaclass.py"
Human adalah Metaclass
```

KESIMPULAN

Berdasarkan ringkasan materi yang telah dibuat, dapat disimpulkan bahwa kelas abstrak merupakan sebuah class yang bersifat abstrak yang berarti kelas abstrak tidak bisa diinstansiasi (tidak bisa dibuat menjadi objek) langsung dan berperan sebagai ‘kerangka dasar’ bagi class turunannya. Kelas abstrak merupakan sebuah kelas yang memiliki satu atau lebih abstract method. Kelas abstrak digunakan untuk membuat struktur logika penurunan dalam proses pemrograman objek. Peggunaan kelas abstrak bertujuan untuk mengimplementasikan sebuah method yang sama ke seluruh class yang diturunkan oleh kelas abstrak. Kelas abstrak dapat digunakan saat ada beberapa fitur umum yang dimiliki oleh semua objek.

Sementara itu, interface adalah koleksi dari method atau fungsi-fungsi yang perlu disediakan oleh implementing class (child class). Interface mengandung metode/fungsi yang didefinisikan sebagai abstract. Metode abstract merupakan sebuah metode yang didefinisikan oleh interface dan interface dapat memberi arti yang konkret pada metode abstract interface. Selain itu, interface dapat digunakan ketika semua fitur perlu diimplementasikan secara berbeda untuk objek yang berbeda. Berbeda dengan interface yang cenderung lebih lambat dibandingkan dengan abstract class, abstract class lebih cepat karena tidak semua implementasi perlu dituliskan pada child class.

Kelas konkret merupakan subclass dari (abstrak) interface yang menyediakan implementasi dari method atau fungsi-fungsi di kelas interface. Kelas konkret perlu dipakai untuk mengimplementasikan interfacenya.

Metaclass adalah tipe kelas spesial yang berfungsi untuk membuat kelas-kelas pada Python Metaclass adalah kelas dari kelas yang dapat mendefinisikan perilaku kelasnya sebagai sebuah turunan dari Metaclass. Metaclass dapat digunakan ketika seseorang ingin mengubah nama kelas yang telah dibuat dalam suatu program.

DAFTAR PUSTAKA

Andersen, A. (1998). A note on reflection in Python 1.5. *Lancaster University, 46.*

Auftechnique. (n.d.). 4 pillar pemrograman berorientasi objek Python. Retrieved April 7, 2023 from https://auftechnique.com/4-pillar-pemrograman-berorientasi-objek/#abstraction

GeeksforGeeks. (2020, Maret 26). *Python-interface module*. Retrieved April 7, 2023 from https://www.geeksforgeeks.org/python-interface-module/

GeeksforGeeks. (2021, Maret 19). *Abstract Classes in Python*. Retrieved April 7, 2023 from https://www.geeksforgeeks.org/abstract-classes-in-python/

Khoirom, S., Sonia, M., Laikhuram, B., Laishram, J., & Singh, T. D. (2020). Comparative Analysis of Python and Java for Beginners. *International Research Journal of Engineering and Technology (IRJET), 7*(8), 4384-4407.

Kumar, B. (2020, Desember 24). *Introduction to Python Interface*. Retrieved April 7, 2023 from PythonGuides.com: https://pythonguides.com/python-interface/

Murphy, W. (n.d.). *Implementing an Interface in Python*. Retrieved April 7, 2023 from Real Python: https://realpython.com/python-interface/

Python. (n.d.). *abc — Abstract Base Classes.* Retrieved April 7, 2023 from https://docs.python.org/3/library/abc.html