filkom.ub.id

DIGITAL TALENT SCHOLARSHIP 2019

Big Data Analytics

Pemrograman Python

Oleh: Imam Cholissodin | imamcs@ub.ac.id, Putra Pandu Adikara, Sufia Adha Putri
Asisten: Guedho, Sukma, Anshori, Aang dan Gusti
Fakultas Ilmu Komputer (Filkom) Universitas Brawijaya (UB)

Pokok Bahasan

1. Fungsi pada Python
2. Perulangan pada Python
3. Objects Oriented pada Python
4. Tugas

Function - Definisi

- Fungsi adalah blok program untuk melakukan tugas tertentu
- Fungsi membuat kode program menjadi *reusable*, artinya hanya didefinisikan sekali saja dan kemudian bisa digunakan berulang kali
- *Modularity* – memecah program besar menjadi sub-sub program
- Sejauh ini, kita sudah menggunakan beberapa fungsi, misalnya fungsi `print()`, `type()`
- Fungsi tersebut adalah fungsi bawaan dari Python
- Kita bisa membuat fungsi kita sendiri sesuai kebutuhan

Function – Mendefinisikan Fungsi

- Kata kunci def diikuti oleh function_name (nama fungsi), tanda kurung dan tanda titik dua (:)
- Parameter/argumen adalah input dari luar yang akan diproses di dalam fungsi
- "function_docstring" bersifat opsional, yaitu sebagai string yang digunakan untuk dokumentasi atau penjelasan fungsi. “function_doctring” diletakkan paling atas setelah baris def.

```
def function_name(parameters):
    """function_docstring"""
    statement(s)
    return [expression]
```

- Setelah itu diletakkan baris – baris pernyataan (statements). Jangan lupa indentasi untuk menandai blok fungsi.
- return bersifat opsional. Gunanya adalah untuk mengembalikan suatu nilai expression dari fungsi

Function – Memanggil Fungsi

```
def sapa(nama):
    """Fungsi ini untuk menyapa seseorang sesuai nama yang dimasukkan
sebagai parameter"""
    print("Hi, " + nama + ". Apa kabar?")

# pemanggilan fungsi
# output: Hi, Umar. Apa kabar?
sapa('Umar')
```

Memanggil Fungsi

- Bila fungsi sudah didefinisikan, maka ia sudah bisa dipanggil dari tempat lain di dalam program.
- Untuk memanggil fungsi caranya adalah dengan mengetikkan nama fungsi berikut paramaternya.

```
sapa('Galih')
>>> Hi, Galih. Apa kabar?

sapa('Ratna')
>>> Hi, Ratna. Apa kabar?
```

Function – Docstring

- Docstring adalah singkatan dari *documentation string*. Ini berfungsi sebagai dokumentasi atau keterangan singkat tentang fungsi yang kita buat. Meskipun bersifat opsional, menuliskan docstring adalah kebiasaan yang baik
- Untuk contoh di atas kita menuliskan docstring. Cara mengaksesnya adalah dengan menggunakan format *namafungsi.__doc__*

```
print(sapa.__doc__)
"""Fungsi ini untuk menyapa
seseorang sesuai nama yang
dimasukkan sebagai parameter"""
```

Function – *Pass by reference vs. value*

- Semua parameter (argument) dalam Bahasa Python menggunakan ***pass by reference***. Artinya mengubah parameter dari suatu fungsi juga akan direfleksikan pada saat pemanggilan fungsi

```
# Function definition is here
def changeme( mylist ):
   "This changes a passed list into this function"
   mylist.append([1,2,3,4]);
   print "Values inside the function: ", mylist
   return

# Now you can call changeme function
mylist = [10,20,30];
changeme( mylist );
print "Values outside the function: ", mylist
```

Jika error saat running: tambahkan tanda kurung di bagian argumen print

Function – Argumen Fungsi

- Pemanggilan fungsi dapat dilakukan dengan menggunakan jenis argumen berikut:

1. Argumen wajib (*required argument*)
2. Argumen *keyword*
3. Argumen *default*
4. Argumen dengan panjang sembarang

Function – *Required Arguments*

- Argumen ditulis dengan urutan posisi yang benar
- Jumlah argumen pada saat pemanggilan fungsi harus sama persis dengan jumlah argumen pada pendefinisian fungsi

```
# Function definition is here
def printme( str ):
   "This prints a passed string into this function"
   print str
   return;

# Now you can call printme function
printme()
```

Output

```
Traceback (most recent call last):
  File "test.py", line 11, in <module>
    printme();
TypeError: printme() takes exactly 1 argument (0 given)
```

Pemanggilan fungsi *printme(),* harus menggunakan satu argumen; jika tidak, akan terjadi *error*

Function – *Keyword Arguments*

- Argumen diambil berdasarkan nama parameternya
- Bisa mengabaikan argumen atau menempatkannya dengan sembarang urutan

```
# Function definition is here
def printinfo( name, age ):
   "This prints a passed info into this function"
   print "Name: ", name
   print "Age ", age
   return;

# Now you can call printinfo function
printinfo( age=50, name="miki" )
```

Output

```
Name:  miki
Age  50
```

Function – *Default Arguments*

- Menggunakan nilai *default* untuk argumen yang tidak diberikan nilainya saat pemanggilan fungsi
- Fungsi akan menampilkan *age* default bila argumen *age* tidak diberikan:

```
# Function definition is here
def printinfo( name, age = 35 ):
   "This prints a passed info into this function"
   print "Name: ", name
   print "Age ", age
   return;

# Now you can call printinfo function
printinfo( age=50, name="miki" )
printinfo( name="miki" )
```

Output

```
Name:  miki
Age  50
Name:  miki
Age  35
```

Function – *Variable-length Arguments*

- Digunakan apabila ingin memproses argumen lebih banyak daripada yang ditentukan pada saat mendefinisikan fungsi

```
def functionname([formal_args,] *var_args_tuple ):
   "function_docstring"
   function_suite
   return [expression]
```

- Tanda asterisk (*) ditulis sebelum nama variabel yang menyimpan nilai dari semua argumen yang tidak didefinisikan
- *Tuple* ini akan kosong bila tidak ada argumen tambahan pada saat pemanggilan fungsi

```
# Function definition is here
def printinfo( arg1, *vartuple ):
   "This prints a variable passed arguments"
   print "Output is: "
   print arg1
   for var in vartuple:
      print var
   return;

# Now you can call printinfo function
printinfo( 10 )
printinfo( 70, 60, 50 )
```

Output

```
Output is:
10
Output is:
70
60
50
```

Function – *Global vs. Local Variables*

- Di Python, tidak semua variabel bisa diakses dari semua tempat. Ini tergantung dari tempat dimana kita mendefinisikan variabel
- Variabel yang didefinisikan dalam fungsi merupakan **local scope.** Hanya bisa diakses didalam fungsi yang mendeklarasikan
- Jika didefinisikan di luar fungsi merupakan **global scope.** Dapat diakses di semua fungsi

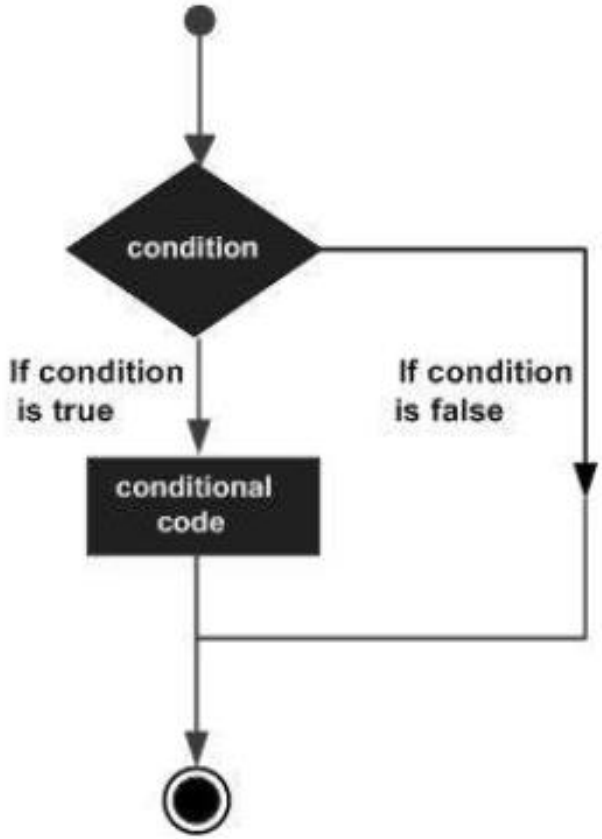
```
total = 0; # This is global variable.
# Function definition is here
def sum( arg1, arg2 ):
   # Add both the parameters and return them."
   total = arg1 + arg2; # Here total is local variable.
   print "Inside the function local total : ", total
   return total;

# Now you can call sum function
sum( 10, 20 );
print "Outside the function global total : ", total
```

Output

```
Inside the function local total :  30
Outside the function global total :  0
```

Python – *Decision Making*

- *Decision Making* – membuat keputusan sesuai dengan kondisi yang terpenuhi
- Percabangan mengevaluasi kondisi atau ekspresi yang hasilnya benar atau salah. Kondisi atau ekspresi tersebut disebut ekspresi Boolean. Hasil dari pengecekan kondisi adalah True atau False.
- Bila benar (True), maka pernyataan yang ada di dalam blok kondisi tersebut akan dieksekusi. Bila salah (False), maka blok pernyataan lain yang dieksekusi



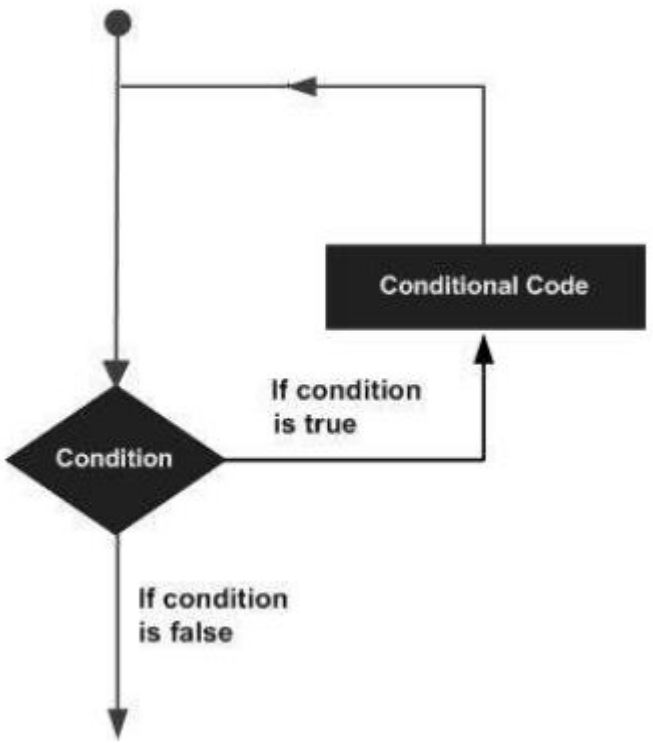
```
var = 100
if ( var == 100 ) : print "Value of expression is 100"
print "Good bye!"
```

```
Value of expression is 100
Good bye!
```

No.	Pernyataan	Deskripsi
1	`if`	Pernyataan if terdiri dari ekspresi boolean diikuti oleh satu baris atau lebih pernyataan.
2	`if…else`	Bila pernyataan if benar, maka blok pernyataan if dieksekusi. Bila salah, maka blok pernyataan else yang dieksekusi.
3	`if…elif…else`	Disebut juga if bercabang. Bila ada kemungkinan beberapa kondisi bisa benar maka digunakan pernyataan if…elif atau if…elif…else

Python – Perulangan (*Loops*)

- Normalnya, *statement* dieksekusi secara sequential (berurutan)
- Bahasa pemograman menyediakan *control structure* untuk memungkinkan jalur eksekusi yang lebih kompleks
- Statement **loop** – mengeksekusi statement beberapa kali



Python – Perulangan (*Loops*)

- Perulangan `for` disebut *counted loop* (perulangan yang terhitung); perulangan `while` disebut *uncounted loop* (perulangan yang tak terhitung)
- `for` biasanya digunakan untuk mengulangi kode yang sudah diketahui banyak perulangannya
- `while` untuk perulangan yang memiliki syarat dan tidak tentu berapa banyak perulangannya

Python – Perulangan for

```
for var in sequence:
    body of for
```

- var adalah variabel yang digunakan untuk penampung sementara nilai dari *sequence* pada saat terjadi perulangan
- *Sequence* adalah tipe data berurut seperti *string*, *list*, dan *tuple*

Python – Perulangan for

- Perulangan terjadi sampai *looping* mencapai elemen terakhir dari *sequence*
- Bila *loop* sudah sampai ke elemen terakhir, maka program akan keluar dari *looping*

```
# Program untuk menemukan jumlah bilangan dalam satu list
# List number
numbers = [7, 5, 9, 8, 4, 2, 6, 4, 1]

# variablel untuk menyimpan jumlah
sum = 0

# iterasi
for each in numbers:
    sum = sum + each

# Output: Jumlah semuanya: 46
print("Jumlah semuanya:", sum)
```

Output

```
Jumlah semuanya: 46
```

Python – Perulangan `while`

```
while expression:
    statement(s)
```

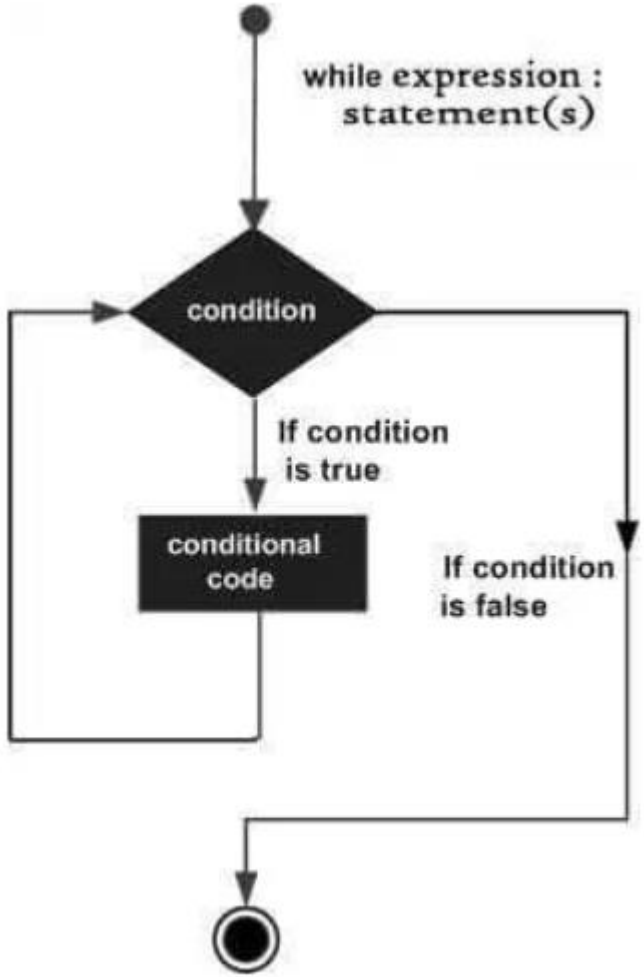
- `statement(s)` bisa terdiri dari satu baris atau satu blok pernyataan
- `Expression` merupakan ekspresi atau kondisi apa saja, dan untuk nilai selain nol dianggap `True`
- Iterasi akan terus berlanjut selama kondisi benar. Bila kondisi salah, maka program akan keluar dari while dan lanjut ke baris pernyataan di luar `while`

Python – Perulangan `while`

```
count = 0
while (count < 5):
    print('The count is:', count)
    count = count + 1
print('Good bye!')
```

Output:

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
Good bye
```



Python – Kendali *Looping*

```
while expression:
    statement(s)
```

- *Looping* umumnya akan berhenti bila kondisi sudah bernilai salah. Akan tetapi, seringkali kita perlu keluar dari *looping* di tengah jalan tergantung keperluan
- Hal ini bisa kita lakukan dengan menggunakan kata kunci `break` dan `continue`

Python – Kendali *Looping*

- Statement break memaksa program keluar dari blok *looping* di tengah jalan
- Sedangkan statement continue menyebabkan program langsung melanjut ke *step* berikutnya dan mengabaikan (*skip*) baris kode di bawahnya (yang satu blok)

```
# contoh penggunaan statement break
for letter in "Programming":
    if letter == "g":
        break
    print("Huruf sekarang:", letter)
print("Good bye")
```

Output

```
Huruf sekarang: P
Huruf sekarang: r
Huruf sekarang: o
Good bye
```

Output yang ditampilkan apabila kode *break* diganti dengan *continue ?*

Python – `while else`

- Python mendukung penggunaan `else` sebagai pasangan dari `while`
- Blok pernyataan `else` hanya akan dieksekusi bila kondisi `while` bernilai salah

```
count = 0
while (count < 5):
    print(count, "kurang dari 5")
    count = count + 1
else:
    print(count, "tidak kurang dari 5")
```

Output

```
0 kurang dari 5
1 kurang dari 5
2 kurang dari 5
3 kurang dari 5
4 kurang dari 5
5 tidak kurang dari 5
```

Python – Object Oriented

- Sejauh ini kita mendesain program berdasarkan fungsi (blok statement yang memanipulasi data) yang dikenal dengan pemograman *procedural approach*
- Python merupakan universal tool untuk pemograman berbasis **object** dan **procedural**
- *Object Oriented Programming*(OOP) merupakan suatu konsep pemrograman yang menekankan pada paradigma atau cara pandang terhadap suatu masalah berdasarkan objek
- Dalam konsep OOP, semua yang ada didunia ini adalah object dan direpresentasikan dalam bentuk objek

Python – Terminologi OOP

1. **Kelas** – Kelas adalah cetak biru atau prototipe dari objek dimana kita mendefinisikan atribut dari suatu objek. Atribut ini terdiri dari data member (variabel) dan fungsi (metode).
2. **Variabel Kelas** – Variabel kelas adalah variabel yang di*share* atau dibagi oleh semua instance (turunan) dari kelas. Variabel kelas didefinisikan di dalam kelas, tapi di luar metode-metode yang ada dalam kelas tersebut.
3. **Data member** – Data member adalah variabel yang menyimpan data yang berhubungan dengan kelas dan objeknya
4. **Overloading Fungsi** – Overloading fungsi adalah fungsi yang memiliki nama yang sama di dalam kelas, tapi dengan jumlah dan tipe argumen yang berbeda sehingga dapat melakukan beberapa hal yang berbeda.
5. **Overloading operator** – Overloading operator adalah pembuatan beberapa fungsi atau kegunaan untuk suatu operator. Misalnya operator + dibuat tidak hanya untuk penjumlahan, tapi juga untuk fungsi lain.

Python – Terminologi OOP

1. **Variabel instansiasi** – Variabel instansiasi adalah variabel yang didefinisikan di dalam suatu metode dan hanya menjadi milik dari instance kelas.
2. **Pewarisan/*Inheritance*** – Inheritansi adalah pewarisan karakteristik sebuah kelas ke kelas lain yang menjadi turunannya.
3. ***Instance*** – *Instance* adalah istilah lain dari objek suatu kelas. Sebuah objek yang dibuat dari prototipe kelas Lingkaran misalnya disebut sebagai instance dari kelas tersebut.
4. **Instansiasi** – Instansiasi adalah pembuatan instance/objek dari suatu kelas
5. **Metode** – Metode adalah fungsi yang didefinisikan di dalam suatu kelas
6. **Objek** – Objek adalah instansiasi atau perwujudan dari sebuah kelas. Bila kelas adalah prototipenya, dan objek adalah barang jadinya.

Python – *Class*

- Beberapa istilah pada konsep pemograman berbasis objek: class, object, attribute, behavior, inheritance dll
- class bisa dianalogikan seperti tubuh dari OOP
- Class merupakan abstraksi atau *blueprint* yang mendefinisikan suatu object tertentu
- Class akan menampung semua attribute dan perilaku dari object tersebut

```
class ClassName:
    'Optional class documentation string'
    class_suite
```

  - *Class* memiliki dokumentasi string yang dapat diakses melalui *ClassName._doc_*
  - *class_suite* terdiri dari semua pernyataan komponen yang mendefinisikan anggota *class,* data atribut, dan *function*

Python – *Class*

- Variabel *empCount* adalah variabel *class* yang nilainya dibagi diantara semua instance dari kelas ini. Dapat diakses sebagai *Employee.empCount* dari dalam *class* atau diluar *class*
- *_init_()* merupakan *methode* pertama (metode khusus), yang disebut *class constructor* atau inisialisasi metode yang dipanggil Python ketika dibuat *instance* baru dari kelas ini
- *methods* yang lain dapat dideklarasikan seperti *function* dengan menambahkan *self* pada argument pertama utnuk setiap *methode*

```
class Employee:
   'Common base class for all employees'
   empCount = 0

   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1

   def displayCount(self):
     print ("Total Employee %d" % Employee.empCount)

   def displayEmployee(self):
      print ("Name : ", self.name,  ", Salary: ", self.salary)
```

Python – Membuat Instance Objects

- Untuk membuat *instances* dari *class,* panggil kelas menggunakan nama *class* dan disertai dengan argument yang diterima metode _init_

```
"This would create first object of Employee class"
emp1 = Employee("Zara", 2000)
"This would create second object of Employee class"
emp2 = Employee("Manni", 5000)
```

- Mengakses *attributes* menggunakan operator dot (.)

```
emp1.displayEmployee()
emp2.displayEmployee()
print ("Total Employee %d" % Employee.empCount)
```

- Output

```
Name :  Zara ,Salary:  2000
Name :  Manni ,Salary:  5000
Total Employee 2
```

Python – Contoh Program Lain

- Jika diperhatikan, dalam class *Car* terdapat 2 attribute *yaitu color = 'black', transmission = 'manual'* dan *method yaitu drive(), reverse().*
- Method dalam konsep OOP mewakili suatu 'behaviour' dari class atau object itu sendiri.

```
class Car:

    color = 'black'
    transmission = 'manual'

def __init__(self, transmission):
    self.transmission = transmission
    print('Engine is ready!')

def drive(self):
    print('Drive')

def reverse(self):
    print('Reverse. Please check your bel
```

Python – *Method*

- Fungsi method dalam konsep OOP adalah untuk merepresentasikan suatu *behavior*
- Dalam contoh di atas suatu object 'mobil' memiliki behaviour antara lain adalah bergerak dan mundur
- Suatu method bisa juga memiliki satu atau beberapa parameter. Sebagai contoh:

```
gear_position = 'N'

def change_gear(self, gear):
    self.gear_position = gear
    print('Gear positiion on: ' + self.gear_position)
```

- Pada method *change_gear()* terdapat 1 parameter yaitu *gear*. Ketika method tersebut dipanggil dan anda tidak memberikan value pada parameter tersebut, maka program akan melempar *error*
- Bagaimanapun juga parameter yang sudah didefinisikan pada suatu method harus memiliki value meskipun *value* tersebut None

Python – Method

- Cara lain yang dapat digunakan adalah dengan mendefinisikan *default value* pada parameter tersebut sejak awal method tersebut dibuat:

```
gear_position = 'N'

def change_gear(self, gear='N'):
    self.gear_position = gear
    print('Gear positiion on: ' + self.gear_position)

self.change_gear()
>>> 'Gear position on: N'
self.change_gear('R')
>>> 'Gear position on: R'
```

- Keyword *self* mengacu pada Class Instance untuk mengakses attribute atau method dari class itu sendiri
- Pemberian keyword *self* pada parameter awal suatu method menjadi wajib jika mendefinisikan *method* tersebut di dalam block suatu class

Python – *Constructor*

- *Method* bernama __init__() disebut dengan *constructor*
- Suatu *constructor* berbeda dengan method lainnya, karena *constructor* akan otomatis dieksekusi ketika membuat *object* dari class itu sendiri
- Suatu *constructor* juga bisa memiliki satu atau beberapa parameter, sama seperti methode pada umumnya namun *constructor* tidak bisa mengembalikan *value*

```
class Car:
    color = 'black'
    transmission = 'manual'

    def __init__(self, transmission):
        self.transmission = transmission
        print('Engine is ready!')

    ...

honda = Car('automatic')
>>> 'Engine is ready!'
```

- Ketika object *honda* dibuat dari class *Car,* constructor langsung dieksekusi. Hal ini berguna jika membutuhkan proses inisialisasi ketika suatu object dibuat

Python – *Object*

- Object merupakan produk hasil dari suatu class
- Jika *class* merupakan *blueprint* dari suatu rancangan bangunan, maka *object* adalah bangunan itu sendiri
- Berikut contoh implementasi dalam bentuk code program:

```
class Car:

    color = 'black'
    transmission = 'manual'
    gear_position = 'N'

    def __init__(self, transmission):
        self.transmission = transmission
        print('Engine is ready!')

    def drive(self):
        self.gear_position = 'D'
        print('Drive')

    def reverse(self):
        self.gear_position = 'N'
        print('Reverse. Please check your behind.')

    def change_gear(self, gear='N'):
        self.gear_position = gear
        print('Gear position on: ' + self.gear_positio

    def get_gear_position(self):
        return self.gear_position

car1 = Car('manual')
car1.change_gear('D-1')

car2 = Car('automatic')
gear_position = car2.get_gear_position()
print(gear_position)
```

Python – Object

- Terdapat 2 buah object *car1* dan *car2* yang dibuat dari class yang sama.
- Masing-masing dari object tersebut berdiri sendiri, artinya jika terjadi perubahan attribute dari object car1 tidak akan mempengaruhi object car2 meskipun dari class yang sama.

```
class Car:

    color = 'black'
    transmission = 'manual'
    gear_position = 'N'

    def __init__(self, transmission):
        self.transmission = transmission
        print('Engine is ready!')

    def drive(self):
        self.gear_position = 'D'
        print('Drive')

    def reverse(self):
        self.gear_position = 'N'
        print('Reverse. Please check your behind.')

    def change_gear(self, gear='N'):
        self.gear_position = gear
        print('Gear position on: ' + self.gear_positio

    def get_gear_position(self):
        return self.gear_position

car1 = Car('manual')
car1.change_gear('D-1')

car2 = Car('automatic')
gear_position = car2.get_gear_position()
print(gear_position)
```

Python – *Inheritance*

```
class Tesla(Car):
    pass    # use 'pass' keyword to define class only

tesla = Tesla()
tesla.drive()
>>> 'Drive'
```

- Salah satu keuntungan dari konsep OOP ialah *reusable codes* yang bisa mengoptimalkan penggunaan code program agar lebih efisien dan meminimalisir redundansi.
- Semua itu berkat adanya fitur *inheritance* yang memungkinkan suatu class (parent) menurunkan semua attribute dan *behavior*-nya ke class (child) lain

Python – *Inheritance*

```
class Tesla(Car):
    pass    # use 'pass' keyword to define class only

tesla = Tesla()
tesla.drive()
>>> 'Drive'
```

- class Tesla merupakan turunan dari class Car
- Jika diperhatikan pada class Tesla tidak didefinisikan method `drive()` namun class tersebut bisa memanggil method `drive()`
- Method tersebut berasal dari class parentnya yaitu class Car, sehingga tidak perlu lagi didefinisikan ulang pada class *child*-nya
- Dengan cara seperti ini anda bisa melakukan *reusable codes* sehingga *source code* menjadi lebih “bersih”

Python – *Method Overriding*

- Ada suatu kondisi dimana suatu method yang berasal dari parent ingin anda modifikasi atau ditambahkan beberapa fitur sesuai kebutuhan pada class child, disinilah peran dari '**overriding** method'.
- Dengan menggunakan fungsi ***super()***, anda bisa memanggil instance dari *class parent* di dalam suatu method untuk memanggil fungsi dari parent tersebut.

```
class Tesla(Car):

    def drive(self):
        super().drive()
        print('LOL Gas')
```

Python – *Private Attribute/Method*

- Tidak semua *attribute* maupun method bisa diturunkan pada class child
- Attribute atau method yang ingin diproteksi agar tidak bisa digunakan pada class turunannya. Dapat dilakukan dengan cara:

```
__factory_number = '0123456789'

def __get_factory_number(self):
    return self.__factory_number
```

Python – *Polymorphism*

- Terakhir dari konsep OOP adalah polimorfisme yang memungkinkan anda untuk membuat banyak bentuk dari satu *object*

```
class Car:
    def fuel(self):
        return 'gas'

class Honda(Car):
    pass

class Tesla(Car):
    def fuel(self):
        return 'electricity'

def get_fuel(car):
    print(car.fuel())

get_fuel(Tesla())
get_fuel(Honda())
>>> 'electricity'
>>> 'gas
```

Python – Atribut Kelas *Built-in*

Setiap kelas di Python memiliki atribut *built-in* (bawaan) yang bisa diakses menggunakan operator titik. Attribut-attribut tersebut adalah sebagai berikut:

- **__dict__** – dictionary yang berisi namespace dari kelas
- **__doc__** – mengakses string dokumentasi (docstring) dari kelas
- **__name__** – nama kelas
- **__module__** – nama modul tempat kelas didefinisikan. Nilai attribut ini di mode interaktif adalah "**__main__**".
- **__bases__** – dasar dari kelas, bila kelas tidak merupakan turunan dari kelas lain, maka induknya dalah kelas object.

Contoh Program

```
# Fig 9.9: fig09_09.py
# Creating a class hierarchy with an abstract base class.

class Employee:
   """Abstract base class Employee"""

   def __init__( self, first, last ):
      """Employee constructor, takes first name and last name.
      NOTE: Cannot create object of class Employee."""

      if self.__class__ == Employee:
         raise NotImplementedError, \
            "Cannot create object of class Employee"

      self.firstName = first
      self.lastName = last

   def __str__( self ):
      """String representation of Employee"""

      return "%s %s" % ( self.firstName, self.lastName )

   def _checkPositive( self, value ):
      """Utility method to ensure a value is positive"""

      if value < 0:
         raise ValueError, \
            "Attribute value (%s) must be positive" % value
      else:
         return value

   def earnings( self ):
      """Abstract method; derived classes must override"""

      raise NotImplementedError, "Cannot call abstract method"
```

Contoh Program

```
36
37  class Boss( Employee ):
38     """Boss class, inherits from Employee"""
39
40     def __init__( self, first, last, salary ):
41        """Boss constructor, takes first and last names and salary"""
42
43        Employee.__init__( self, first, last )
44        self.weeklySalary = self._checkPositive( float( salary ) )
45
46     def earnings( self ):
47        """Compute the Boss's pay"""
48
49        return self.weeklySalary
50
51     def __str__( self ):
52        """String representation of Boss"""
53
54        return "%17s: %s" % ( "Boss", Employee.__str__( self ) )
55
```

Contoh Program

```
56  class CommissionWorker( Employee ):
57     """CommissionWorker class, inherits from Employee"""
58
59     def __init__( self, first, last, salary, commission, quantity ):
60        """CommissionWorker constructor, takes first and last names,
61        salary, commission and quantity"""
62
63        Employee.__init__( self, first, last )
64        self.salary = self._checkPositive( float( salary ) )
65        self.commission = self._checkPositive( float( commission ) )
66        self.quantity = self._checkPositive( quantity )
67
68     def earnings( self ):
69        """Compute the CommissionWorker's pay"""
70
71        return self.salary + self.commission * self.quantity
72
73     def __str__( self ):
74        """String representation of CommissionWorker"""
75
76        return "%17s: %s" % ( "Commission Worker",
77           Employee.__str__( self ) )
78
```

Contoh Program

```
79  class PieceWorker( Employee ):
80     """PieceWorker class, inherits from Employee"""
81
82     def __init__( self, first, last, wage, quantity ):
83        """PieceWorker constructor, takes first and last names, wage
84        per piece and quantity"""
85
86        Employee.__init__( self, first, last )
87        self.wagePerPiece = self._checkPositive( float( wage ) )
88        self.quantity = self._checkPositive( quantity )
89
90     def earnings( self ):
91        """Compute PieceWorker's pay"""
92
93        return self.quantity * self.wagePerPiece
94
95     def __str__( self ):
96        """String representation of PieceWorker"""
97
98        return "%17s: %s" % ( "Piece Worker",
99           Employee.__str__( self) )
100
```

Contoh Program

```
101 class HourlyWorker( Employee ):
102    """HourlyWorker class, inherits from Employee"""
103
104    def __init__( self, first, last, wage, hours ):
105       """HourlyWorker constructor, takes first and last names,
106       wage per hour and hours worked"""
107
108       Employee.__init__( self, first, last )
109       self.wage = self._checkPositive( float( wage ) )
110       self.hours = self._checkPositive( float( hours ) )
111
112    def earnings( self ):
113       """Compute HourlyWorker's pay"""
114
115       if self.hours <= 40:
116          return self.wage * self.hours
117       else:
118          return 40 * self.wage + ( self.hours - 40 ) *\
119            self.wage * 1.5
120
121    def __str__( self ):
122       """String representation of HourlyWorker"""
123
124       return "%17s: %s" % ( "Hourly Worker",
125          Employee.__str__( self ) )
126
```

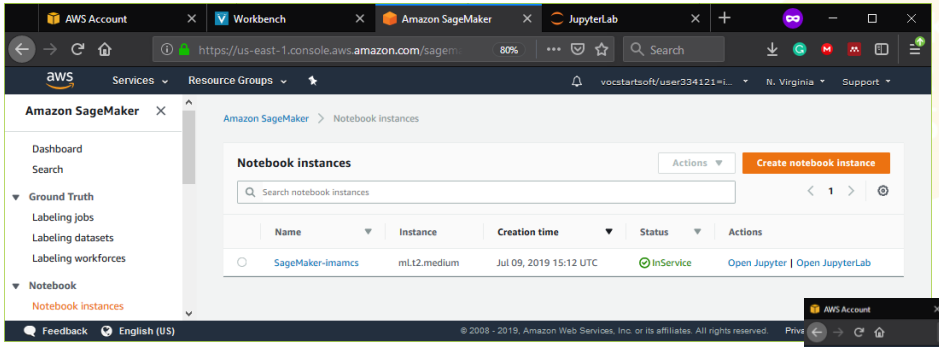
Contoh Program

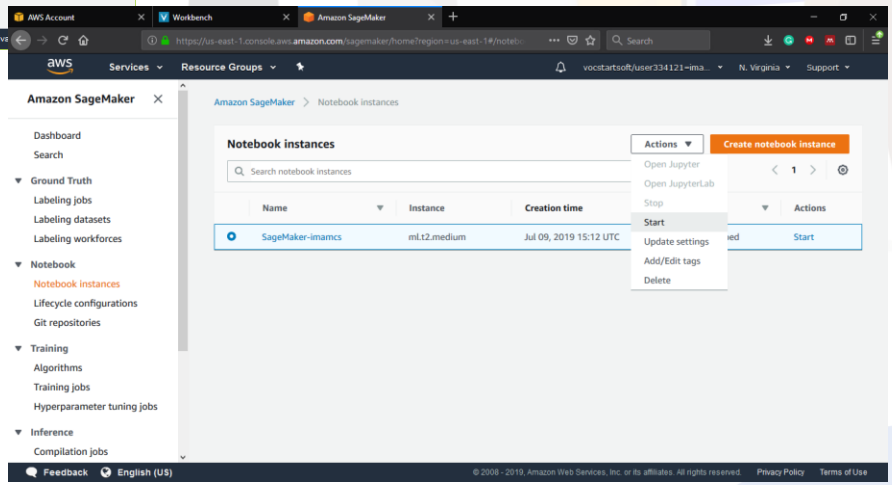
```
127 # main program
128
129 # create list of Employees
130 employees = [ Boss( "John", "Smith", 800.00 ),
131               CommissionWorker( "Sue", "Jones", 200.0, 3.0, 150 ),
132               PieceWorker( "Bob", "Lewis", 2.5, 200 ),
133               HourlyWorker( "Karen", "Price", 13.75, 40 ) ]
134
135 # print Employee and compute earnings
136 for employee in employees:
137    print "%s earned $%.2f" % ( employee, employee.earnings() )
```

```
            Boss: John Smith earned $800.00
Commission Worker: Sue Jones earned $650.00
     Piece Worker: Bob Lewis earned $500.00
    Hourly Worker: Karen Price earned $550.00
```

Start Jupyter Notebook

- Silahkan menggunakan Jupyter yang anda install *from scratch* di minggu pertama (di EC2) atau menggunakan SageMaker yang baru dibuat sebelumya. Dan mohon dicek credit anda, jika menggunkan SageMaker terlalu menguras credit Anda, sebaiknya di-*stop* setiap kali selesai menggunakan (jgn lupa selalu backup file *.ipynb ke github Anda atau di local).





Latihan langsung di Kelas Ke-1 & Pembahasan
Link kode “http://bit.ly/2mBYz0e”

Silahkan dicoba dijalankan dengan Jupyter notebook yang Anda buat sebelumnya di Ubuntu 16.04 atau dengan SageMaker notebook (JupyterLab) yang baru Anda buat hari ini.

Lab-Sesi9-1

Membaca Berkas di Python

Pada notebook ini Anda akan mempelajari cara membaca berkas (*file*) pada bahasa Python. Pada akhir lab ini, Anda diharapkan mampu membaca berkas teks.

Daftar Isi

- Download Data
- Membaca Berkas Teks
- Cara Membaca Berkas yang Lebih Baik

Perkiraan waktu pengerjaan: **40 min**

Download Data

```
In [0]: # Download Example file
        !wget -O Example1.txt https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/PY0101EN/labs/example1.txt
```

Lab-Sesi9-2

Menulis dan Menyimpan Berkas di Python

Pada notebook ini Anda akan mempelajari cara menulis ke berkas teks pada bahasa Python. Di akhir lab, Anda diharapkan dapat menulis berkas dan menyalin berkas.

Daftar Isi

- Menulis Berkas
- Menyalin Berkas

Perkiraan waktu pengerjaan: **15 menit**

Menulis Berkas

Penulisan berkas diawali dengan proses membuka berkas menggunakan fungsi `open()` dengan parameter 'w' dilanjutkan dengan fungsi `write()`. Fungsi `write()` memerlukan parameter masukan berupa string.

```
In [0]: # Menulis string ke berkas

        with open('Example2.txt', 'w') as writefile:
            writefile.write("This is line A")
```

Latihan langsung di Kelas Ke-2 & Pembahasan

- Buatlah program untuk mencari rata-rata dari nilai tugas-tugas setiap siswa, serta mencari nilai terkecil dan nilai terbesar dari nilai keseluruhan

Tugas Individu

1. Buatlah rangkuman materi di atas dengan cara berikut:
   - Dari file *.ipynb, berikan penjelasan tambahan lalu *convert* ke pdf (atau cukup dengan pindahkan screenshot kode ke *.doc/x, lalu berikan penjelasan dari koding yg anda buat, lalu convert ke *.pdf) yang refer dari “Latihan langsung di Kelas Ke-1 dan Latihan langsung di Kelas Ke-2”.

lalu simpan dalam satu file PDF dengan nama file, misal
“[Nama Lengkap Mhs]-[Pert. Ke-2.1/..]”

Dan dari semua tugas dalam 1 minggu di-merger, dengan nama file seperti:
“[Nama Lengkap Mhs]-[Minggu Ke-1/2/..]”, lalu cek plagiasi diturnitin dari hasil merger tersebut
> Register ke turnitin
> Masukkan **id class**: 21563495 & **enrool key**: filkomub9302

filkom.ub.ac.id

DIGITAL TALENT SCHOLARSHIP 2019

Big Data Analytics

Terimakasih

Oleh: Imam Cholissodin | imamcs@ub.ac.id, Putra Pandu Adikara, Sufia Adha Putri
Asisten: Guedho, Sukma, Anshori, Aang dan Gusti
Fakultas Ilmu Komputer (Filkom) Universitas Brawijaya (UB)

Tugas Individu

- Mengerjakan Review Questions Cognitiveclass Module 3 pada cognitiveclass.ai (Dibuktikan dengan Screenshot hasil yang sudah *di-convert* ke pdf) --> Optional

Materi Tambahan

- Menyelesaikan course “PY0101EN” pada cognitiveclass.ai --> Optional
  - Module 3 Lab – Conditions and Branching
  - Module 3 Lab – Loops
  - Module 3 Lab – Functions
  - Module 3 Lab – Objects and Classes