Fakultas Informatika
School of Computing
Telkom University

CII1A3- PENGENALAN PEMROGRAMAN

PERTEMUAN KE-6

FUNGSI

PREVIEW MATERI

- Decomposition and Abstraction
- Function, Parameter, Argument
- Variable Scope
- Python Built-in Functions

Decomposition and Abstraction

Apa yang sudah kita pelajari sejauh ini, yaitu dari output sampai looping, membuat kita mampu membuat **program-program dengan skala kecil**, anggaplah program yang hanya membutuhkan 10 baris atau 20 baris kode. Sekarang bayangkan jika kita ingin membangun sebuah website e-commerce, sebuah game, atau membangun sebuah operating system sendiri, maka tentu dibutuhkan banyak sekali baris kode, mungkin **ribuan, puluhan ribu, atau ratusan ribu baris kode**. Sebagai gambaran, Windows Operating sistem memiliki sekitar 50 juta baris kode.

Coba bayangkan....

Anggaplah kita ingin membangun aplikasi untuk pencatatan transaksi di kasir sebuah toko atau minimarket, aplikasi ini biasa disebut dengan **Point of Sales (POS)**. Aplikasi ini akan menangani:

1. Pencatatan inventory yang terdiri dari penambahan dan pengurangan stok, termasuk mencatat detil tiap produk misalnya kode, nama, kategori, harga.
2. Melakukan pencatatan transaksi jual beli yang terdiri dari perhitungan total bayar, perhitungan diskon jika ada, dan pengurangan stok barang.
3. Membuat laporan mingguan, bulanan, atau tahunan untuk melihat laba rugi, barang yang paling laris, barang yang sering dibeli bersamaan, dan sebagainya.
4. Manajemen pengguna, yang minimal terdiri dari kasir dan manager/pemilik toko, di mana tiap jenis pengguna punya akses atau fitur yang berbeda.
5. Pembayaran pembelian dari website e-commerce, tiket kereta, voucher game, dan lain-lain.

Kita bisa terus menambah fitur dari aplikasi tersebut.

bagaimana anda memulai meng-*coding* program besar di atas?

Decomposition

Secara natural, jika kita melihat sesuatu yang **kompleks** maka kita akan bertanya, bagaimana **menyederhanakannya**, atau **membagi-bagi permasalahan tersebut ke dalam bagian-bagian kecil** yang lebih sederhana? Konsep tersebut biasa disebut dengan ***decomposition***.

Dalam pemrograman, **decomposition adalah memecah-mecah kode program menjadi beberapa sub-program yang lebih kecil**, salah satunya melalui penggunaan **function**. Kode program yang besar dipisah dan dikelompokkan berdasarkan fungsi spesifik yang dikerjakan bagian kode tersebut.

Dari kasus di slide sebelumnya maka akan dibuat function → → →

- function login
- function input_stock
- function hitung_diskon
- function total_bayar
- function beli
- function urut_produk dan lain sebagainya..

Abstraction

Dalam pemrograman sudah menjadi hal yang biasa bahwa kita biasanya **menggunakan kode atau fungsi yang dibuat oleh orang lain** untuk membangun suatu program yang lebih besar secara lebih cepat dan lebih mudah. Begitu juga fungsi yang anda buat, bukan tidak mungkin bisa digunakan oleh orang lain, misalnya karena anda **bekerja di dalam tim** dimana setiap orang bertugas terhadap fitur yang berbeda, namun saling berkaitan, sehingga beberapa **fungsi anda akan digunakan oleh orang lain begitu juga sebaliknya**.

Untuk bisa menggunakan fungsi orang lain, yang anda perlu ketahui adalah:

- **bagaimana cara menggunakan fungsi tersebut,**
- **bagaimana efek dari penggunaan fungsi tersebut.**

Sedangkan bagaimana detil susunan kode (implementasi) di dalam fungsi tersebut tidak perlu anda ketahui.

Mekanisme atau cara-cara untuk menyembunyikan detail yang kompleks dan hanya menunjukkan informasi sederhana yang perlu diketahui pengguna disebut dengan **abstraction**.

Abstraction

Di dalam pemrograman, abstraction didapatkan salah satunya melalui **function**, dimana saat menggunakan function yang telah dibuat, **programmer hanya perlu memberikan input saja, tanpa perlu mengetahui bagaimana proses di dalam function tersebut**.

Melanjutkan contoh POS di atas, seandainya anda bekerja di dalam tim dimana tiap fungsi dikerjakan oleh programmer yang berbeda, maka perlu didefinisikan hal-hal yang menjadi input dan output fungsi agar dapat digunakan dengan tepat oleh orang lain, misalnya:

- function login membutuhkan input berupa username dan password, dan outputnya adalah True jika berhasil dan False jika username dan password tidak tepat.

Function

Function adalah satu group statement yang melakukan **tugas spesifik tertentu**. Function bertujuan **men-*decompose*** program menjadi **bagian-bagian yang lebih kecil**. Function membuat kode program yang besar menjadi lebih terorganisir sehingga lebih mudah dikelola. Selain itu, function dapat **menghindari repetisi penulisan kode yang sama** berulang kali.

Secara umum terdapat dua jenis function:

- **User-defined function**: yaitu function yang dibuat oleh programmer untuk mendukung program yang sedang dikerjakannya.
- **Built-in function**: yaitu function yang sudah tersedia dan siap digunakan setelah kita menginstall python.

Pendefinisian Function

Kapanpun kita merasa bahwa ada bagian kode tertentu **akan digunakan kembali**, maka itu adalah tanda bahwa mungkin kita perlu membuatnya ke dalam function. Atau ketika kita **kesulitan untuk mengingat apa yang bagian kode tertentu lakukan**, atau **mencari bagian kode yang melakukan suatu tugas tertentu**, maka itu juga sinyal bahwa kita perlu membuat fungsi.

cara untuk mendefinisikan sebuah fungsi di python:

```
def <nama_function>(<parameter1>, <parameter2>, ...):
    """docstring"""
    <body function>
    return <expression>
```

Penjelasan di slide selanjutnya

- **keyword def** adalah penanda mulainya header sebuah fungsi.
- **<nama_function>** mengikuti aturan seperti penamaan variable, sebaiknya dipilih nama yang intuitif sesuai isi fungsi tersebut
- **<parameter1>, <parameter2>,** dst disebut dengan parameter fungsi, dan sifatnya adalah opsional. Parameter adalah variable yang digunakan untuk memberikan nilai yang dibutuhkan oleh fungsi.
- **( )**, walaupun parameter adalah opsional, namun buka dan tutup kurung bersifat wajib
- **"""docstring"""** bersifat **opsional**, merupakan documentation string yang menjelaskan tentang fungsi tersebut, apa yang dilakukan oleh fungsi, parameter yang dibutuhkan dan nilai yang dikembalikan oleh fungsi.
- **<body function>** seperti halnya body while, if, dan body program secara umum
- **return <expression>** bersifat **opsional**. Jika ada, maka begitu baris return dieksekusi, maka fungsi akan selesai (walaupun masih ada statement di bawahnya) dan nilai setelah return akan menjadi nilai hasil fungsi tersebut. **Jika tidak ada return, nilai default fungsi adalah None**.

Contoh Penggunaan Fungsi


```python
# fungsi menampilkan menu makanan
def tampil_menu():
    print('Menu 1: Udang Bakar')
    print('Menu 2: Ayam Goreng')
    print('Menu 3: Cumi Rebus')
```

- **tampil_menu** adalah nama fungsi
- tidak ada parameter fungsi, namun () tetap harus ditulis
- body fungsi terdiri dari 3 buah statement print

```python
# fungsi untuk menghitung luas lingkaran
def hitung_luas_lingkaran(r):
    luas =  3.14 * r**2
    return luas
```

- **hitung_luas_lingkaran** adalah nama fungsi
- **r** adalah sebuah **parameter**, yang dalam hal ini adalah radius lingkaran, dibutuhkan fungsi agar bisa menghitung luas lingkaran
- **return luas**, maka nilai fungsi adalah nilai yang tersimpan dalam variable luas

```python
# fungsi untuk memprint dengan tambahan dekorasi tertentu
def my_print(jlh, kata):
    """
    Fungsi untuk memprint kata dengan jumlah tertentu.

    Parameters:
    jlh (integer) : kata akan di print sebanyak jlh kali
    kata (string) : kata yang akan diprintkan
    """
    print("Kita akan print sebanyak", jlh, "kali")
    print("-"*30)
    for i in range(jlh):
        print("Cinta", kata)
    print("-"*30)
```

- terdapat 2 buah parameter
- terdapat **docstring** yang menjelaskan parameter fungsi
- tidak terdapat **return**, karena my_print tidak menghitung suatu nilai, melainkan melakukan print berdasarkan format tertentu

Pemanggilan Function

```
nama_function(argument)
```

dimana **argument** adalah nilai yang diberikan ke parameter fungsi.

Dalam bahasa pemrograman lain atau referensi lain, ada juga yang menyebutkan parameter sebagai **formal parameter**, dan argument sebagai **actual parameter**.

Namun jika **fungsi memiliki return**, maka biasanya nilai fungsi akan digunakan lebih lanjut, sehingga pemanggilan fungsi dikombinasikan misalnya dengan **print**, assignment ke variable lain, atau dalam expression.

```
x = nama_function(argument) # nilai function di-assign ke variable
print(nama_function(argument)) # nilai function di-print
z = 100 + nama_function(argument) # nilai function digunakan dalam expression
```

Pemanggilan fungsi tanpa return value

```
tampil_menu()
```

```
Menu 1: Udang Bakar
Menu 2: Ayam Goreng
Menu 3: Cumi Rebus
```

Terlihat bahwa daftar menu terprint di console. Hal ini tentunya akan memudahkan jika di dalam program kita harus menampilkan menu beberapa kali di bagian kode yang berbeda-beda.

Pemanggilan fungsi dengan return value

```
luas = hitung_luas_lingkaran(10)
print(luas)
```

```
314.0
```

```
print(hitung_luas_lingkaran(10) + hitung_luas_lingkaran(20))
```

```
1570.0
```

Nilai Default Fungsi

Fungsi tampil_menu() tidak memiliki return value, kira-kira jika fungsinya di print, apa yang akan tertampil?

```
x = tampil_menu()
print("Nilai fungsi :", x)
```

```
Menu 1: Udang Bakar
Menu 2: Ayam Goreng
Menu 3: Cumi Rebus
Nilai fungsi : None
```

Hal ini membuktikan bahwa jika fungsi tidak memiliki return value, maka nilai defaultnya adalah **None**.

Kesesuaian jumlah dan urutan antara argumen dengan parameter

Jika fungsi memiliki parameter, maka saat pemanggilannya fungsi harus diberikan argumen sesuai dengan jumlah parameter tersebut. Jika kurang atau lebih, maka akan terjadi error. Perhatikan contoh berikut ini:

SESUAI

```
my_print(10, "halo")
```

```
Kita akan print sebanyak 10 kali
------------------------------
Cinta halo
Cinta halo
Cinta halo
Cinta halo
Cinta halo
Cinta halo
Cinta halo
Cinta halo
Cinta halo
Cinta halo
------------------------------
```

TIDAK SESUAI

```
my_print("satu", 10)
```

```
Kita akan print sebanyak satu kali
------------------------------
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-90-52b8b79eceeb> in <module>()
----> 1 my_print("satu", 10)

<ipython-input-84-aa2285fd6e41> in my_print(jlh, kata)
     10     print("Kita akan print sebanyak", jlh, "kali")
     11     print("-"*30)
---> 12     for i in range(jlh):
     13         print("Cinta", kata)
     14     print("-"*30)

TypeError: 'str' object cannot be interpreted as an integer
```

Lokasi Pemanggilan Function

Fungsi dapat dipanggil di dalam program, di dalam fungsi lain, atau pun di dalam python console.

Perhatikan contoh berikut ini dimana fungsi dipanggil dari dalam fungsi lainnya:

```
def hitung_volume_tabung(r, t):
    return hitung_luas_lingkaran(r) * t
```

```
print(hitung_volume_tabung(10, 10))
```

```
3140.0
```

Docstring

Dokumentasi tentang parameter dan return value dari fungsi sangat penting sebagai bagian dari konsep ***abstraction***, yaitu **pengguna fungsi** tidak perlu tau bagaimana implementasi/kode fungsi secara detail, namun cukup mengetahui apa maksud fungsi tersebut, dan bagaimana cara menggunakannya. Hal ini akan sangat bermanfaat ketika kita bekerja di dalam tim, di mana tiap orang mengerjakan fungsi yang berbeda-beda namun saling menggunakan fungsi yang dikerjakan orang lain. Bahkan, bisa juga fungsi yang kita buat di-*share* ke public dan bisa digunakan oleh siapa saja.

Di python, cara untuk membuat dokumentasi fungsi adalah melalui *document string* atau *docstring*. Sementara di bahasa lain juga tersedia dengan sebutan yang berbeda, misalnya di java disebut *document comments*.

```python
# fungsi untuk memprint dengan tambahan dekorasi tertentu
def my_print(jlh, kata):
    """
    Fungsi untuk memprint kata dengan jumlah tertentu.

    Parameters:
    jlh (integer) : kata akan di print sebanyak jlh kali
    kata (string) : kata yang akan diprintkan
    """
    print("Kita akan print sebanyak", jlh, "kali")
    print("-"*30)
    """
    ini bukan docstring
    namun tetap tidak dianggap sebagai kode yang harus dieksekusi
    """
    for i in range(jlh):
        print("Cinta", kata)
    print("-"*30)
```

- **# fungsi** untuk memprint ... adalah komentar, namun **bukan docstring**
- docstring **dimulai dan ditutup** dengan tanda **""" (triple double quote)** dan **harus berada tepat di bawah header fungsi**.
- tidak seperti komentar, **docstring menjadi sebuah properti/bagian yang melekat pada fungsi tersebut**, dan bisa diakses melalui nama_fungsi.__doc__ atau help(nama_fungsi).

Penggunaan Docstring

```
print(my_print.__doc__)
```

```
    Fungsi untuk memprint kata dengan jumlah tertentu.

    Parameters:
    jlh (integer) : kata akan di print sebanyak jlh kali
    kata (string) : kata yang akan diprintkan
```

```
help(my_print)
```

```
Help on function my_print in module __main__:

my_print(jlh, kata)
    Fungsi untuk memprint kata dengan jumlah tertentu.

    Parameters:
    jlh (integer) : kata akan di print sebanyak jlh kali
    kata (string) : kata yang akan diprintkan
```

Terlihat bahwa docstring bisa sangat membantu bagi orang yang ingin menggunakan fungsi, tanpa harus membuka *source code* fungsi tersebut. Untuk itu, ada baiknya kita membiasakan diri untuk menuliskan *docstring* untuk fungsi-fungsi yang kita buat.

Variable Scope

Scope pada pemrograman (tidak hanya Python) adalah tentang lokasi di mana saja suatu entitas (e.g., variable) bisa diakses. Istilah ***local variable*** dan ***global variable*** adalah contoh yang terkait dengan scope variable. Pemahaman tentang scope sangat penting untuk memastikan tidak ada conflict antar variable dan menghindari terjadinya *bug* pada program.

Global Variable

Variable global adalah variable yang dideklarasikan di luar fungsi atau di program utama. Variable yang memiliki scope global bisa diakses di luar dan di dalam fungsi.

```
x = 10

def cetak():
    print("dalam fungsi:", x)

cetak()
```

```
dalam fungsi: 10
```

Variable x memiliki scope global karena didefinisikan di dalam program utama, sehingga x bisa diakses dari dalam fungsi cetak.

Keyword global

Jika kita memang ingin mengubah variable global dari dalam fungsi, maka perlu menambahkan keyword global seperti contoh di bawah ini:

```
x = 10

def tambah():
    global x # maka x yang dimaksud dalam fungsi ini adalah x global, meskipun ada x di-ass
    y = x + 1
    x = y

tambah()
print("setelah fungsi:", x)
```

```
setelah fungsi: 11
```

```
x = 10
tambah()
print(x)
```

```
11
```

Local Variable

Variable yang dideklerasikan di dalam fungsi akan memiliki scope local, sehingga tidak bisa diakses dari luar fungsi tersebut.

```
def foo():
    z = 10 # variable lokal, tidak bisa diakses dari luar fungsi foo()

foo()
print("setelah fungsi:", z) # z tidak bisa diakses dari luar fungsi
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-1e039e1178d6> in <module>()
      3
      4 foo()
----> 5 print("setelah fungsi:", z) # z tidak bisa diakses dari luar fungsi

NameError: name 'z' is not defined
```

Namun, variable local fungsi bisa diakses fungsi lain yang didefinisikan di dalam fungsi tersebut. Perhatikan contoh di bawah ini:

```
def foo():
    local_foo = 10
    def bar():
        print("variable local_foo dari dalam bar:", local_foo)

    bar()

foo()
```

```
variable local_foo dari dalam bar: 10
```

Pass by Value vs Pass by Reference

Konsep berikutnya yang perlu dipahami tentang fungsi adalah bagaimana nilai dari *argument* diberikan ke *parameter* saat fungsi dipanggil. 2 konsep yang populer adalah **pass by value** dan **pass by reference**.

Jika kita menyimpan sebuah data di memori, misalnya melalui assign sebuah variable a = 10, maka ada 2 hal yang perlu kita ketahui:

- **Value**: data atau nilai yang disimpan di memory
- **Reference**: alamat memori yang menyimpan data tersebut

Sedangkan variable a seperti nama alias untuk memudahkan kita dalam memprogram sehingga tidak perlu mengingat alamat memori. 2 variable selain bisa menyimpan value yang sama, juga bisa menyimpan alamat yang sama. Hal ini diatur oleh bahasa pemrograman masing-masing.

Pass by Value vs Pass by Reference

Pass by Value: data **di-*copy*** dari yang memanggil ke fungsi yang dipanggil. Sehingga meskipun datanya sama, namun **di memori letaknya berbeda**. Akibatnya, ketika data **diubah di dalam fungsi**, **data di yang memanggil tetap tidak berubah**.

Pass by Reference: yang diberikan oleh pemanggil ke fungsi yang dipanggil adalah **alamat memori data**, sehingga alamat memori yang ditunjuk oleh fungsi dan yang memanggil adalah sama. Akibatnya, jika data **dirubah di dalam fungsi**, maka **data tersebut juga berubah di yang memanggil fungsi**.

Python tidak memiliki pengaturan agar suatu parameter *pass by value* atau *pass by reference*. Python hanya menganut aturan ***pass by object reference*** atau ada juga yang menyebutnya ***pass by assignment***. Untuk memahaminya, mari kita lihat lebih dalam cara kerja assignment pada Python melalui contoh berikut ini:

```
a = 10
print('id a:',id(a))
b = a
print('id b:',id(b))
b = 20
print('id b:',id(b))
b = a
print('id b:',id(b))
```

```
id a: 10914784
id b: 10914784
id b: 10915104
id b: 10914784
```

id() adalah built-in function yang akan mengembalikan identitas unik suatu objek. Terlihat **b = a** menyebabkan kedua variable menyimpan identitas yang sama. Identitas di python bisa dianggap seperti reference atau alamat suatu objek di memory. Sehingga bisa dipahami bahwa assignment di python adalah pemberian reference suatu objek ke sebuah variable.

Ketika b di-assign kembali dengan nilai yang lain, maka identitasnya berubah.

Sekarang kita lihat contoh menggunakan list berikut ini:

```
list_a = [10]
print('id list_a:',id(list_a))
list_b = list_a
print('id list_b:',id(list_b))
list_b = [20]
print('id list_b:',id(list_b))
list_b = list_a
print('id list_b:',id(list_b))
```

```
id list_a: 139815337496136
id list_b: 139815337496136
id list_b: 139815327721800
id list_b: 139815337496136
```

Terlihat bahwa hasilnya konsisten seperti contoh sebelumnya, yaitu assignment memberikan reference objek, serta identitas variable list_b akan berubah ketika di-assign kembali dengan data yang lain, karena data baru tersebut memiliki alamat memori yang berbeda.

Namun, terdapat perbedaan pada kedua jenis data tersebut, antara int dan list. Perhatikan contoh berikut ini:

```
list_a = [10]
list_b = list_a
print('list_b sebelum modifikasi :', list_b)
list_b[0] = 20
print('list_b setelah modifikasi :', list_b)
print('list_a setelah modifikasi list_b :', list_a)
```

```
list_b sebelum modifikasi : [10]
list_b setelah modifikasi : [20]
list_a setelah modifikasi list_b : [20]
```

Terlihat bahwa data pada list_b dapat diubah tanpa harus meng-assign kembali list_b dengan data baru, melainkan perubahan tersebut dilakukan melalui update element list.
Perhatikan:

- list_b = [20] : ini adalah assignment kembali variable list_b
- list_b[0] = 20 : variable list_b tidak di-assign kembali, yang di-assign kembali adalah elemen list_b di index 0.

Sedangkan pada variable int, tidak ada cara untuk bisa mengubah data tersebut selain dengan assign kembali, yang artinya mengubah referensi variable sehingga berbeda dari sebelumnya.

Prinsip *assignment* tersebut digunakan pada pemanggilan fungsi sehingga seolah-olah terjadi proses *assignment* di samping:

```
parameter = argument
```

Berdasarkan penjelasan cara kerja assignment pada list dan int, maka bisa dipahami jika parameter memiliki tipe int, efeknya akan seperti pass by value. Artinya, jika parameter tersebut diubah nilainya di dalam fungsi, maka pasti hanya dapat dilakukan dengan assignment kembali, yang mengakibatkan berbedanya referensi parameter dengan referensi argument yang pada akhirnya perubahan tersebut tidak akan berdampak pada yang memanggil fungsi.

Sedangkan jika parameter bertipe list, maka akan seperti pass by reference selama perubahan data pada list tidak dilakukan dengan assignment kembali.

```
def func_update_list(list_x):
    list_x[0] = list_x[0] + 1
    print('dalam fungsi :', list_x)

list_1 = [10, 20]
func_update_list(list_1)
print('setelah panggil fungsi :', list_1)
```

```
dalam fungsi : [11, 20]
setelah panggil fungsi : [11, 20]
```

```
def func_update_list(list_x):
    list_x = [11, 20]
    print('dalam fungsi :', list_x)

list_1 = [10, 20]
func_update_list(list_1)
print('setelah panggil fungsi :', list_1)
```

```
dalam fungsi : [11, 20]
setelah panggil fungsi : [10, 20]
```

Mutable dan Immutable Object

Berdasarkan kemampuan mengubah data, maka suatu object di python dapat digolongkan ke dalam ***mutable object*** dan ***immutable object***.

Mutable objek adalah objek yang bisa diubah setelah dibuat, dengan kata lain value yang disimpan di suatu variable bisa diubah tanpa melakukan ***reassignment*** variable.

Berdasarkan tipe data yang sudah kita pelajari sejauh ini, berikut pembagiannya:

- immutable object: int, float, str, bool
- mutable object: list

Untuk penjelasan lengkap dapat melihat python docs:
https://docs.python.org/3/reference/datamodel.html

Fakultas Informatika
School of Computing
Telkom University

TERIMA KASIH