INSTITUT
TEKNOLOGI
HARAPAN
BANGSA
Veritas vos liberabit

MODUL PRAKTIKUM

IF – 2P2
Praktikum Matematika Informatika

Ria Chaniago, S.T., M.T.

Budi Kurniawan, S.T.

DAFTAR ISI

V. FUNCTION

Tujuan dan sasaran pembelajaran:

1) Mahasiswa dapat memahami konsep dasar *function* pada Python.
2) Mahasiswa dapat mengimplementasikan suatu *function* dalam memecahkan berbagai masalah.

V.1 *Overview*

Pada setiap pembuatan program yang kompleks dan memiliki banyak fitur didalamnya, maka seorang *programmer* diharuskan menggunakan *function* dalam membuat kode dari fitur-fitur tersebut. Setidaknya ada dua alasan mengapa seorang *programmer* dituntut untuk bisa membuat kode dalam suatu *function*.

Pertama adalah ***reusability***. Ketika suatu *function* sudah didefinisikan, maka *function* tersebut dapat digunakan berulang kali. Anda dapat memanggil *function* tersebut berulang kali dalam program yang dibuat, sehingga Anda tidak perlu menuliskan baris program yang sama terus menerus. Tentu saja kelebihan ini menghemat waktu Anda dalam membuat suatu program.

Kedua adalah ***abstraction***. Jika Anda hanya ingin menggunakan suatu *function* pada program Anda, maka Anda tidak perlu tahu cara kerja yang sebenarnya dari *function* tersebut. Anda hanya perlu tahu bagaimana cara kerja suatu *function*, jika dan hanya jika ketika Anda sedang membuat *function* tersebut atau ketika Anda ingin mengubah cara kerja dari *function* tersebut.

V.2 Cara Membuat *Function* pada Python

Berikut adalah sintaks penulisan dari suatu *function* pada Python.

```
def function_name(arguments):
    expression(1)
    expression(2)
    expression(3)
    .
    .
    expression(n)
    return [expression] # optional
```

```
# contoh pembuatan function
def cetak_nama(nama):
    print('Hai,', nama)
    return

def tambah(bilangan1, bilangan2):
    hasil = bilangan1 + bilangan2
    return hasil

# cara memanggil function
cetak_nama('Hermione')
tambah(1, 2)
```

V.3 Jenis-Jenis *Arguments*

Function pada Python dapat dipanggil dengan menggunakan beberapa jenis parameter (*arguments)* sebagai berikut.

V.3.1 *Positional Arguments*

Positional arguments adalah *function* memanggil parameter yang digunakan berdasarkan urutan dari parameter tersebut.

```
def bagi(bilangan1, bilangan2):
    hasil = bilangan1 / bilangan2
    return hasil

# panggil function bagi()
bagi(1, 2)
bagi(2, 1)

# output: bagi(1, 2) != bagi(2, 1)
0.5
2.0
```

V.3.2 *Keyword Arguments*

Keyword arguments adalah *function* memanggil parameter yang digunakan berdasarkan kata kunci (*keyword*) dari parameter tersebut.

```
def bagi(bilangan1, bilangan2):
    hasil = bilangan1 / bilangan2
    return hasil
```

```
# panggil function bagi()
bagi(bilangan1 = 1, bilangan2 = 2)
bagi(bilangan2 = 2, bilangan1 = 1)

# output:
0.5
0.5
```

Pada dasarnya *keyword arguments* adalah *positional arguments* dengan memiliki nilai *default*. Oleh karena itu, pada saat memanggil *keyword arguments function*, letakan *positional arguments* sebelum *keyword arguments*.

V.3.3 *Default Arguments*

Default arguments adalah parameter bernilai *default* yang digunakan jika terdapat suatu parameter yang tidak tersedia ketika suatu *function* dipanggil.

```
def cetak_informasi(nama, usia = 23):
    print("Nama =", nama)
    print("Usia =", usia)
    return

# panggil function cetak_informasi()
cetak_informasi('Hermione')
cetak_informasi("Harry Potter", usia = 25)

# output:
Nama = Hermione
Usia = 23

Nama = Harry Potter
Usia = 25
```

Syarat yang harus dipenuhi untuk membuat *default arguments* adalah letakan semua *non-default arguments* sebelum *default arguments*.

V.3.4 *Variable-length Arguments*

Ketika suatu *function* diproses, terkadang diperlukan lebih banyak parameter dibandingkan definisi awal dari *function* tersebut (seperti contoh di bawah ini, ada saatnya seseorang memiliki banyak sekali hobi). Untuk mengatasi permasalahan ini, maka Python menyediakan metode

variable-length arguments. Salah satu ciri dari *variable-length arguments* adalah dengan menggunakan tanda asterisk (*) diawal nama variabel.

```
def cetak_hobi(nama, *tuple_hobi):
      print("Nama =", nama)
      total_hobi = len(tuple_hobi)
      if total_hobi > 0:
            print("Hobi =", tuple_hobi)
      else:
            print("Tidak punya hobi")

      return

# panggil function cetak_hobi()
cetak_hobi('Ron')
cetak_hobi('Hermione', 'Membaca', 'Makan', 'Travelling')

# output:
Nama = Ron
Tidak punya hobi

Nama = Hermione
Hobi = ('Membaca', 'Makan', 'Travelling')
```

Syarat yang harus dipenuhi untuk *variable-length arguments* adalah letakan *variable-length arguments* setelah *non-keyword arguments*, dan parameter setelah *variable-length arguments* harus berupa *keyword arguments*.

V.4 *Return Statement*

Return statement merupakan sintaks agar keluar dari *function* yang dibuat, yang secara opsional dapat mengembalikan suatu nilai ke program yang memanggil *function* tersebut. Apabila tidak ada *argument* setelah sintaks *return* (contoh pada *function* `cetak_nama()` pada upabab V.2), maka bisa dikatakan *function* tersebut tidak mengembalikan suatu nilai atau biasa disebut prosedur (sintaksnya bisa juga ditulis `return None`).

```
def tambah(bilangan1, bilangan2):
    hasil = bilangan1 + bilangan2
    print("Hasil di dalam function =", hasil)
    return hasil
```

```
# panggil function tambah()
hasil = tambah(1, 2)
print("Hasil di luar function =", hasil)

# output:
Hasil di dalam function = 3
Hasil di luar function = 3
```

Function pada Python dapat mengembalikan (*return*) lebih dari satu nilai. Berikut contohnya:

```
def foo():
    return True, False

# memanggil function update_list()
value1, value2 = foo()
print(value1)
print(value2)

# output
True
False
```

V.5 *Pass by Reference* dan *Pass by Value*

Seperti yang sudah pernah dibahas sebelumnya, bahwa pada Python (dan juga bahasa pemrograman yang lain) terdapat *mutable object* dan *immutable object*. Pada Python sendiri yang termasuk *mutable object* adalah *list, dict, set, byte array*. Sedangkan yang termasuk *immutable object* adalah *int, float, complex, string, tuple, frozen set* [catatan: *immutable version of set*], *bytes*. Pemahaman tentang *mutable* dan *immutable object* akan sangat berguna dalam membuat *function*, sehingga *function* yang dibuat dapat meningkatkan efisiensi memori.

Pengertian *pass by reference* adalah ***an alias or reference to the actual parameter is passed to the method***. Sedangkan pengertian *pass by value* adalah ***the method parameter values are copied to another variable and then the copied object is passed***.

Perhatikan contoh *function* di bawah ini:

```
def update_list(mylist):
    print("Lokasi memori sebelum di-update di dalam function =",id(my_list))
    mylist.append(7)
    print("Lokasi memori sesudah di-update di dalam function =",id(my_list))
    return
```

```
# memanggil function update_list()
my_list = [1,2]
print(my_list)
print("Lokasi memori sebelum di-update di luar function =",id(my_list))
update_list(my_list)
print(my_list)
print("Lokasi memori sesudah di-update di luar function =",id(my_list))

# output
[1, 2]      # nilai sebelum function dipanggil
Lokasi memori sebelum di-update di luar function = 47593232
Lokasi memori sebelum di-update di dalam function = 47593232
Lokasi memori sesudah di-update di dalam function = 47593232
[1, 2, 7]   # nilai sesudah function dipanggil
Lokasi memori sesudah di-update di luar function = 47593232
```

Potongan program di atas merupakan contoh dari *pass by reference*. Dapat dilihat bahwa variabel `my_list` yang dipakai memiliki lokasi memori yang sama di alamat `47593232`, meskipun sudah melakukan *update* nilai. Agar tidak mengubah nilai dari parameter yang digunakan, bisa menggunakan variabel pembantu untuk menampung nilai dari parameter, perhatikan *function* `update_list()` di bawah ini:

```
def update_list(mylist):
    temp = list(mylist)    # temp adalah variabel pembantu
    temp.append(7)
    return temp

# memanggil function update_list()
mylist = [0]
print(mylist)
updated_mylist = update_list(mylist)
print(mylist)
print(updated_mylist)

# output
[0]
[0]
[0, 7]
```

Sekarang perhatikan contoh *function* lain di bawah ini:

```
def update_number(number):
    print("Lokasi memori sebelum di-update di dalam function =",id(number))
    number += 1
    print(number)
    print("Lokasi memori sesudah di-update di dalam function =",id(number))
    return

# memanggil function update_number()
number = 6
print(number)
print("Lokasi memori sebelum di-update di luar function =",id(number))
update_number(number)
print(number)
print("Lokasi memori sesudah di-update di luar function =",id(number))

# output
6              # nilai sebelum function dipanggil
Lokasi memori sebelum di-update di luar function = 1542510848
Lokasi memori sebelum di-update di dalam function = 1542510848
7              # nilai di dalam function
Lokasi memori sesudah di-update di dalam function = 1542511008
6              # nilai sesudah function dipanggil
Lokasi memori sesudah di-update di luar function = 1542510848
```

Potongan program di atas merupakan contoh dari *pass by value*. Dapat dilihat bahwa variabel `number` yang dipakai berbeda lokasi memorinya sesudah di-*update* nilainya, yaitu dari alamat `1542510848` menjadi `1542511008`.

Bahan Diskusi

Sering terjadi perdebatan mengenai *pass by reference* dan *pass by value* pada setiap bahasa pemrograman, termasuk pada Python juga. Perhatikan potongan program di bawah ini:

```
def update_list(mylist):
        mylist.append(1)
        mylist = [2, 3]

my_list = [0]
update_list(my_list)
print(my_list)
```

Menurut Anda apa hasil dari `print(my_list)`? Apakah itu *pass by reference*? Atau *pass by value*?

V.6 Variabel Global dan Variabel Lokal

Dalam pemrograman Python, *programmer* harus mengetahui apa itu variabel global dan variabel lokal. Variabel global adalah variabel yang bisa diakses dari semua *function*, sedangkan variabel lokal adalah variabel yang hanya bisa diakses di dalam *function* tempat variabel tersebut berada. Jadi Python mulai mencari variabel lokal terlebih dahulu, lalu kemudian variabel global.

```
var = 'global'

def foo():
    var = 'lokal'
    print(var)

print(var)
foo()

# output
global
lokal
```

V.7 Fungsi Rekursif

Fungsi rekursif dalam pemrograman merupakan fungsi yang memanggil dirinya sendiri. Gambar di bawah ini bisa menggambarkan apa itu fungsi rekursif.

Dalam membuat suatu fungsi rekursif ada dua hal yang harus diperhatikan:

1) Langkah basis. Langkah basis adalah kondisi dimana fungsi rekursif akan berhenti.
2) Langkah rekursif. Langkah rekursif adalah langkah yang akan terus dieksekusi selama fungsi rekursif berjalan.

Berikut adalah contoh dari fungsi rekursif dalam memecahkan masalah mencari nilai faktorial.

```
def faktorial(n):
    if n == 1:
       return 1                    # langkah basis
    else:
       return n * faktorial(n-1)   # langkah rekursif

# memanggil function faktorial()
hasil = faktorial(5)
print(hasil)

# output
120
```

Cara kerja dari fungsi rekursif untuk faktorial dapat dijabarkan sebagai berikut:

faktorial(5) = 5 * faktorial(4)

faktorial(4) = 4 * faktorial(3)

faktorial(3) = 3 * faktorial(2)

faktorial(2) = 2 * faktorial(1)

faktorial(1) = 1

Maka faktorial(5) = 5 * 4 * 3 * 2 * 1 = 120