

Basic Programming :

Iteration

Array and Other Data Types

Function

Learning Progress Review Week 6

By:
MARVEL TEAM

Fikrie | Natalia | Satria

Materi yang dipelajari

1. Iteration pada Python

- ❑ Pengenalan Iteration
- ❑ Flowchart of conditions
- ❑ While Loop
- ❑ For Loop
- ❑ Nested Loop

2. Array and Other Data Types

- ❑ Array
- ❑ Non-primitive Data Types
- ❑ Tuple
- ❑ List Dua Dimensi
- ❑ Dictionary
- ❑ Nested Dictionary

3. Function pada Python

- ❑ Fundamental Functional Programming
- ❑ Mengapa Function berguna?
- ❑ Function hands on : Syntax Function, Multiple parameters, Global vs Local Variable, Nested Function
- ❑ Pengenalan Library dalam Python
- ❑ String manipulation

1. *Iteration* pada Python

Iterations merupakan suatu program yang menjalankan perintah dalam bentuk proses yang berulang-ulang

Pengenalan *Iteration*

Iterations merupakan suatu program yang menjalankan perintah dalam bentuk proses yang berulang-ulang, yang dilakukan bertujuan untuk men-generate atau menciptakan suatu output tertentu yang kita inginkan.

Di dalam *Iterations* terdapat 2 tipe yaitu:

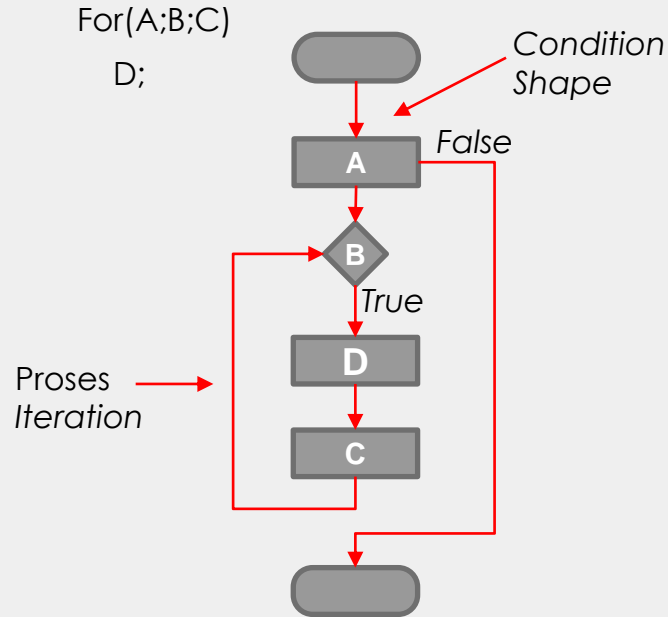
- **Definite Iteration**

Jumlah dari repitisi atau perulangan yang dilakukan itu sudah di spesifikasikan sejak awal.

- **Indefinite Iteration**

akan terus di lakukan *count block*-nya di eksekusi sampai memenuhi suatu kondisi tertentu.

Flowchart of Condition



Condition Shape:

For {subject} **in** {iterable thing}:
{you guys take an action}

??

or

While {condition is true}:
{you guys keep in actions}

Quite Clear

While Loop

While Loop adalah perulangan uncountable atau perulangan yang jumlah proses pengulangannya tidak di tentukan. **While** akan menjalankan baris kode di dalam blok kodenya secara terus menerus selama masih memenuhi ekspresi yang sudah di tentukan sebelumnya.

Rumus **While**:

```
while ekspresi:
    statement
    statement
```

Contoh pengaplikasikannya:

```
a = 1
while a < 6:
    print(a)
    a += 1
```

1
2
3
4
5

For Loop

For Loop merupakan perulangan yang cara kerjanya menyerupai *while loop*, hanya saja **for** lebih digunakan dalam perulangan yang sudah diketahui jumlah perulangannya (*countable*).

Di dalam Python perulangan **For** lebih sering digunakan untuk memproses array atau himpunan.

Rumus **For**:

```
▶ var = [a, b]
for i in var:
    statement
    statement
```

Contoh pengaplikasikannya:

```
team_marvel = ['Muhammad Fikrie', 'Natalia Ringo', 'Satria Triputra']

for name in team_marvel:
    print('Nama Team MARVEL: {}'.format(name))

Nama Team MARVEL: Muhammad Fikrie
Nama Team MARVEL: Natalia Ringo
Nama Team MARVEL: Satria Triputra
```

Nested Loop

Nested Loop atau *loop* bersarang, Bahasa pemrograman python mengizinkan penggunaan *loop* di dalam *loop*.

Di dalam Python perulangan *nested loop* lebih sering digunakan saat suatu project sudah memasuki ke level kompleks

Rumus **Nestedr**:

```
▶ for i in var:  
    for j in i:  
        statement  
        statement
```

Contoh pengaplikasikannya:

```
▶ angka = int(input('silahkan masukkan angka: '))  
bar = 1  
  
while bar < angka + 1:  
    bar_1 = 1  
    while bar_1 < bar + 1:  
        print(bar_1, end=' ')  
        bar_1 += 1  
    print('\n')  
    bar += 1
```


2. *Array and Other Data Types*

Array adalah tipe data terstruktur

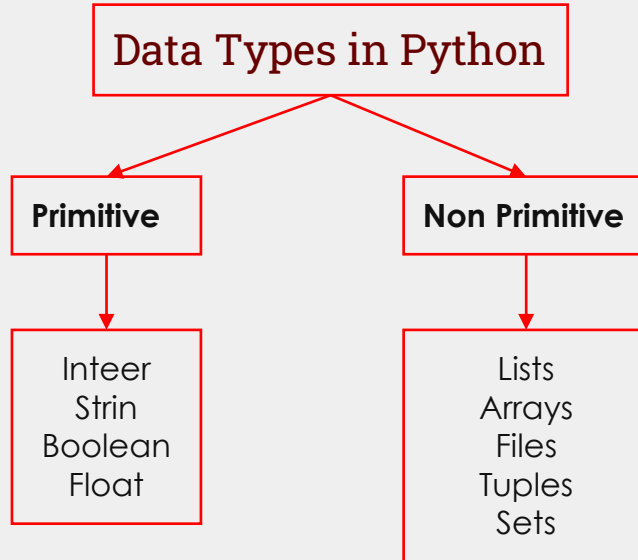
Array

Array merupakan tipe data terstruktur yang berfungsi untuk menyimpan sejumlah data yang bertipe sama. Data di dalamnya disebut elemen. Tiap elemen memiliki nilai yang disebut index untuk menandai ada di urutan ke berapakah elemen tersebut.

- Elemen pertama adalah 0.
- Sehingga jumlah elemen $N - 1$.

Di Python, *main built array* untuk *data structure* adalah List.

Non-primitive Data Types



List adalah objek yang digunakan untuk menyimpan elemen-elemen yang heterogen.

Dictionary bersifat *unordered*, tidak memiliki index, tetapi mempunyai key untuk mengakses elemen di dalam *dictionary*, serta bersifat *Immutable*.

Array

Membuat Array

→

```
[ ] variabel = [value, value, value]
```

```
[ ] city = ['Saint Petersburg', 'Krasnoyarsk', 'Sochi', 'Vladivostok']
```

Menampilkan Elemen

→

```
[2] city = ['Saint Petersburg', 'Krasnoyarsk', 'Sochi', 'Vladivostok']  
print(city[2])
```

Sochi

Menampilkan Panjang Array

→

```
[3] city = ['Saint Petersburg', 'Krasnoyarsk', 'Sochi', 'Vladivostok']  
print(len(city))
```

4

Pengulangan Array

→

```
[4] number = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]  
for x in number:  
    if x % 2 == 0:  
        print(x)
```

2
4
6
8
10
12

Array

Mengganti Elemen



```
[6] city = ['Saint Petersburg', 'Krasnoyarsk', 'Sochi', 'Vladivostok']  
city[2] = 'Volgograd'  
print(city)
```

```
['Saint Petersburg', 'Krasnoyarsk', 'Volgograd', 'Vladivostok']
```

Slicing (mengambil elemen tertentu)



```
[8] city = ['Saint Petersburg', 'Krasnoyarsk', 'Sochi', 'Vladivostok']  
print(city[1:3])
```

```
['Krasnoyarsk', 'Sochi']
```

Menambah elemen (paling akhir)



```
[9] city = ['Saint Petersburg', 'Krasnoyarsk', 'Sochi', 'Vladivostok']  
city.append('Murmansk')  
print(city)
```

```
['Saint Petersburg', 'Krasnoyarsk', 'Sochi', 'Vladivostok', 'Murmansk']
```

Menambah elemen (posisi yang ditarget)



```
[18] number = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]  
number.insert(2, 'x')  
print(number)
```

```
[1, 2, 'x', 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

Array

Penggabungan List



```
[20] city = ['Saint Petersburg', 'Krasnoyarsk', 'Sochi', 'Vladivostok']  
      city2 = ['Chelyabinsk', 'Murmansk', 'Omsk']  
      cities = city + city2  
      print(cities)
```

```
['Saint Petersburg', 'Krasnoyarsk', 'Sochi', 'Vladivostok', 'Chelyabinsk', 'Murmansk', 'Omsk']
```

Menghapus Elemen



```
[22] city = ['Saint Petersburg', 'Krasnoyarsk', 'Sochi', 'Vladivostok', 'Chelyabinsk', 'Murmansk', 'Omsk']  
      city.remove('Vladivostok')  
      print(city)
```

```
['Saint Petersburg', 'Krasnoyarsk', 'Sochi', 'Chelyabinsk', 'Murmansk', 'Omsk']
```

Tuple

Tuple merupakan data berkelompok yang nilai setiap elemen dan jumlah elemennya konstan (tidak dapat diubah maupun ditambah/dikurangi).

Membuat Tuple



```
[ ] kota1 = ('Moskow', 12.5)
    kota2 = ('Saint Petersburg', 5.3)
    kota3 = ('Novosibirsk', 1.6)
    kota4 = ('Yekaterinburg', 1.5)
    kota5 = ('Nizhny Novorod', 1.2)
```

Mengakses Elemen



```
[23] kota1 = ('Moskow', 12.5)
    kota2 = ('Saint Petersburg', 5.3)
    kota3 = ('Novosibirsk', 1.6)
    kota4 = ('Yekaterinburg', 1.5)
    kota5 = ('Nizhny Novorod', 1.2)
    print(kota1[0])
```

Moskow

Tuple

Membuat List



```
[24] kota1 = ('Moskow', 12.5)
      kota2 = ('Saint Petersburg', 5.3)
      kota3 = ('Novosibirsk', 1.6)
      kota4 = ('Yekaterinburg', 1.5)
      kota5 = ('Nizhny Novorod', 1.2)
      kota = [kota1, kota2, kota3, kota4, kota5]
      print(kota)

[('Moskow', 12.5), ('Saint Petersburg', 5.3), ('Novosibirsk', 1.6), ('Yekaterinburg', 1.5), ('Nizhny Novorod', 1.2)]
```

Mengakses Elemen dari List



```
[29] kota = [('Moskow', 12.5), ('Saint Petersburg', 5.3), ('Novosibirsk', 1.6), ('Yekaterinburg', 1.5), ('Nizhny Novorod', 1.2)]
      for penduduk in kota:
          print(penduduk[1])

12.5
5.3
1.6
1.5
1.2
```


List Dua Dimensi

Membuat List



```
[30] halo = [[7, 8, 9], [77, 88, 99], [777, 888, 999]]  
print(halo)
```

```
[[7, 8, 9], [77, 88, 99], [777, 888, 999]]
```

**Mengakses
Elemen dari
List**



```
[31] halo = [[7, 8, 9], [77, 88, 99], [777, 888, 999]]  
print(halo[1][2])
```

```
99
```

Dictionary

Dictionary merupakan data yang menyimpan beberapa nilai dengan skema **key:value**, dimana 1 key akan memiliki 1 value. Karakteristik tipe data ini mirip dengan tipe data *list* yaitu, bisa diakses dan panjangnya dinamis.

Dictionary



```
[ ] v = {  
    <key> : <value>  
    <key> : <value>  
    .  
    .  
    .  
    <key> : <value>  
}
```

Dictionary

Membuat Dictionary



```
[33] Indonesia = {  
    'Merdeka' : 17081945,  
    'Bentuk Negara' : 'Republik',  
    'Ibukota' : 'Jakarta',  
    'Kepala Negara' : 'Presiden',  
    'Kepala Pemerintahan' : 'Presiden',  
    'Keanggotaan PBB' : True  
}  
print(type(Indonesia))  
  
<class 'dict'>
```

Memanggil Nilai



```
[34] Indonesia = {  
    'Merdeka' : 17081945,  
    'Bentuk Negara' : 'Republik',  
    'Ibukota' : 'Jakarta',  
    'Kepala Negara' : 'Presiden',  
    'Kepala Pemerintahan' : 'Presiden',  
    'Keanggotaan PBB' : True  
}  
print(Indonesia['Merdeka'])
```

17081945

Nested Dictionary

Membuat Nested Dictionary



```
[35] country = {  
    'spain': { 'capital':'madrid', 'population':46.77 },  
    'france': { 'capital':'paris', 'population':66.03 },  
    'germany': { 'capital':'berlin', 'population':80.62 },  
    'norway': { 'capital':'oslo', 'population':5.084 },  
    'indonesia' : { 'capital':'jakarta', 'population':250 }  
}
```

```
for x, y in country.items():  
    print('Populasi ' + x + ' adalah ' + str(y['population']))  
    print('Ibukota ' + x + ' adalah ' + y['capital'])
```

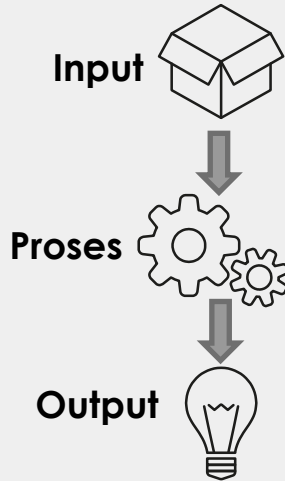
```
↳ Populasi spain adalah 46.77  
Ibukota spain adalah madrid  
Populasi france adalah 66.03  
Ibukota france adalah paris  
Populasi germany adalah 80.62  
Ibukota germany adalah berlin  
Populasi norway adalah 5.084  
Ibukota norway adalah oslo  
Populasi indonesia adalah 250  
Ibukota indonesia adalah jakarta
```

3. *Function* pada Python

Function seperti pabrik yang terdiri dari input, proses, dan output

Fundamental *Functional Programming*

Seperti **pabrik**



Functional programming adalah paradigma *programming* yang mana secara umum *software* akan dikumpulkan dalam bentuk suatu proses sistem *input* dan *output* melalui eksekusi *script code function*-nya.

Function merupakan salah satu *tools* dalam Python yang dapat membantu mengerjakan suatu perintah secara berulang tanpa menuliskan kembali *script code*-nya.

Function adalah suatu sistem yang menerima *input* dan akan memprosesnya dalam suatu *code* yang didefinisikan dalam *body function*-nya, kemudian mengeluarkan *output*. Secara sederhana, **function seperti pabrik yang mempunyai *input*, urutan proses dan *output*.**

Mengapa *Function* berguna?

Manfaat *Function* antara lain :

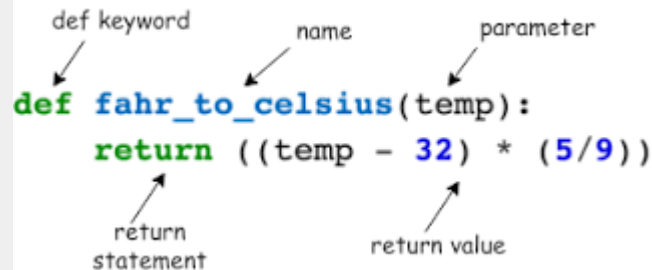
1. Ketika kita mempunyai proses berulang, kita dapat menggunakan *function* untuk menyederhanakan code.
2. Dapat memisahkan setiap *workflow* dalam *function* tersebut. Contohnya, Clean data, Call data from csv, etc.
3. *Function* akan memudahkan kita fokus dalam menyelesaikan suatu masalah daripada melakukan *debugging* atau uji coba *running* berkali-kali dengan proses yang sama.
4. Setiap masalah atau *error* akan mudah dilacak (*trace*) menggunakan *function*.
5. Relatif sangat bermanfaat dalam mengurangi memori.

Syntax Function

Syntax function dalam bahasa Python adalah :

- Diawali dengan keyword **def**
- Diikuti dengan nama fungsi dan parameter yang diperlukan
- Selanjut, fungsi akan mengembalikan hasil pemrosesan tersebut melalui *return statement*.

Contoh pengaplikasiannya:



```
def fahr_to_celsius(temp):
    return ((temp - 32) * (5/9))
```

```
def cube(x):
    value = x **3
    return value
```

```
tigapangkattiga = cube(3)
print(tigapangkattiga)
```

27

Perbedaan Parameter dan *Argument*

Parameter berada dalam suatu function setelah nama fungsinya. Sedangkan, **argument** merupakan nilai yang dilempar kepada fungsi untuk diproses melalui baris kode di dalam fungsi tersebut.

The diagram shows a Python function definition and a function call. A speech bubble labeled "Parameters" points to the variables 'a' and 'b' in the function definition. Another speech bubble labeled "Arguments" points to the values '2' and '3' in the function call.

```
# Function Definition
def add(a, b):
    return a + b

# Function Call
add(2, 3)
```

Parameters

Arguments

getktool

Docstring

Docstring berguna untuk :

- Mendeskripsikan apa yang dilakukan suatu fungsi
- Sebagai dokumentasi
- Diletakkan dalam baris setelah *function header*

Contoh pengaplikasiannya:

```
def square(value):  
    """Return the square of a value"""  
    new_value = value ** 2  
    return new_value
```


Function header

Docstring

Multiple Parameters dalam Function

Contoh pengaplikasiannya:

Multiple parameters



```
def raise_to_power (value1, value2):  
    """Raise value1 to the power of value2"""  
    new_value = value1 ** value2  
    return new_value
```

```
raise_to_power(2, 3)
```

```
8
```

Global vs Local Variable

Global variable dapat diakses menyeluruh di dalam sebuah program dari awal sampai akhir dengan menggunakan *function*. Pada dasarnya, *global variable* berada di luar *function* dan tetap berada di dalam memori Python kecuali kita menghapusnya atau menutup program Python kita.

Local variable hanya dapat diakses di dalam *function* dimana mereka didefinisikan/dideklarasikan. Setelah *script code function* selesai, maka *local variable* telah ada.

```
x = 1

def foo():
    x = 10
    y = 20

for i in range(1, 2):
    x = 100
    y = 200
```

global namespace variable

local namespace variables

local namespace variables

Nested Function

Nested function adalah fungsi yang didefinisikan dalam fungsi lain. Dalam Python, fungsi ini memiliki fungsi langsung ke *variable* dan nama-nama yang didefinisikan dalam suatu fungsi tertutup.

Contoh pengaplikasiannya:

```
def mod2plus5(x1, x2, x3):
    """Returns the remainder plus 5 of three values"""

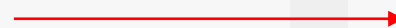
    new_x1 = x1 % 2 + 5
    new_x2 = x2 % 2 + 5
    new_x3 = x3 % 2 + 5

    return new_x1, new_x2, new_x3
```

```
mod2plus5(10, 21, 30)
```

```
(5, 6, 5)
```

Disederhanakan menjadi :



```
def mod2plus5(x1, x2, x3):
    """Returns the remainder plus 5 of three values"""

    def inner(x):
        """Returns the remainder plus 5 of a value"""
        return x % 2 + 5

    return inner(x1), inner(x2), inner(x3)
```

```
mod2plus5(10, 21, 30)
```

```
(5, 6, 5)
```

Pengenalan *Library* dalam Python

Library merupakan sekumpulan kode yang memiliki fungsi – fungsi tertentu dan dapat dipanggil ke dalam program lain. *Library* dibuat oleh seseorang, komunitas atau suatu perusahaan. Dengan *library*, hampir 80% pekerjaan pemrograman telah selesai, dan *programmer* dapat fokus dalam penyelesaian masalah.

Beberapa *library* yang terkenal dalam Data Science adalah NumPy, pandas, sql-alchemy, matplotlib, seaborn, scikit-learn, tensorflow, dll.



Penggunaan *Library* dalam Python

Untuk menggunakan *library* dalam Python, diperlukan *syntax* untuk memasukkan *library* tersebut ke dalam *workspace* kita. *Syntax* yang digunakan adalah **import**.

Contoh :

```
import math  
import datetime
```

Math Library dalam Python

Math library merupakan *library* yang digunakan untuk melakukan operasi matematika yang lebih kompleks, seperti akar, logaritma, pangkat, dll. Sebagian besar fungsi akan mengembalikan nilai dengan tipe data *float*. Beberapa fungsi di dalam *math library* antara lain :

```
#### fungsi log
logaritma = math.log(100)
print(logaritma)
```

```
4.605170185988092
```

```
#### fungsi log10
print(math.log10(100))
```

```
2.0
```

```
#### fungsi sqrt
print(math.sqrt(16))
```

```
4.0
```

```
#### contoh round
print(round(logaritma, 2))
```

```
4.61
```


Datetime Library dalam Python

Datetime library merupakan *library* yang digunakan untuk memanipulasi tipe tanggal waktu dalam Python.

Beberapa fungsi di dalam *datetime library* antara lain :

```
### contoh now
import datetime as dt
dt.datetime.now()
```

```
datetime.datetime(2021, 6, 10, 12, 46, 39, 617607)
```

```
### contoh strptime untuk mengubah string ke datetime
freedom = '17/08/1945'
freedom2 = dt.datetime.strptime(freedom, '%d/%m/%Y')
print(freedom2)
```

```
1945-08-17 00:00:00
```

```
### contoh strftime untuk mengubah suatu datetime ke string
date = dt.datetime.strptime(freedom2, '%d %B, %Y')
print(date)
```

```
17 August, 1945
```

```
### contoh timedelta untuk menghitung beda waktu antara dua datetime
date_now = dt.datetime.now()
freedom_age = date_now - freedom2
print(freedom_age)
print(type(freedom_age))
print(freedom_age.days)
```

```
27691 days, 13:09:12.083844
<class 'datetime.timedelta'>
27691
```

String Manipulation

Memmanipulasi string dalam Data Science adalah kemampuan (*skill*) yang sangat penting untuk dimiliki karena seringkali data yang diterima dalam kondisi yang tidak rapi. Tipe data *string* Python bisa diproses sebagai *array* dari karakter.

Contoh :

```
### membuat string dan mengakses elemennya
string1 = "Hari ini hari Sabtu"
print(string1[5])
```

i

```
# menampilkan string dalam huruf kecil
print(string1.lower())
```

```
hari ini hari sabtu
```

```
### mendapatkan panjang string
print(len(string1))
```

19

```
### split string
```

```
print(string1.split(" "))
```

```
['Hari', 'ini', 'hari', 'Sabtu']
```

```
### strip
```

```
string2 = " Besok hari Minggu "
print("Panjang String (Sebelum): {}".format(len(string2)))
print(string2.strip())
print("Panjang String (Setelah): {}".format(len(string2.strip())))
```

```
Panjang String (Sebelum): 19
```

```
Besok hari Minggu
```

```
Panjang String (Setelah): 17
```

Special Thanks to :



Slide template by SlideCarnival