

[ .py ]



Written by  
**Rakuu x Amamiya Ren**

# Table of Contents

<b>Intro</b>	<b>5</b>
<b>[ Beginning ]</b>	<b>6</b>
1. Tipe Data dan Operator	6
a. Tipe Data Basic	6
b. Tipe Data String	8
c. Operator Basic	9
d. Prioritas Operator	10
e. Built in Function	10
2. Variable dan Input	11
a. Deklarasi Variable	11
b. Variable Assignment	11
c. Compound Assignment Operators	12
d. Operator Aritmatika	12
e. Input Function	13
3. Logical and Comparison Operators	14
a. Comparison Operators	14
b. Comparison Operator Is & In	14
c. Logical Operators	15
4. Conditional Statement	16
a. Selection Structure Concept	16
b. If Statement Syntax	16
c. Indentation	16
d. Pass Keyword	17
e. Elif Statement	17
f. If-Else Statement	17
g. Nested Conditional Statement	18
5. Loop	18
a. Basic Loop - For Loop Syntax	18
b. Sequence Object Iteration	19
c. While Loop Syntax	19
d. Break vs Continue Keywords	19

e. Nested Loop	20
6. Sequence Data Types	21
a. Mutable vs Immutable Data Types	21
b. List - Declaration	22
c. List - Indexing	22
d. List - Addition and Deletion	23
e. List - Slicing	25
f. List - In Operator	26
g. Tuple - Basic	26
h. Sequences Data Types - Operator	27
i. Sequences Data Types - Count Method	28
j. Sequences Data Types - Operators	29
7. Dictionary	29
a. Basic	29
b. Key - Value Pair	30
c. Adding, Modifying, Delete Values	30
d. Function and Operators	31
e. JSON Datatypes - Basics	32
f. JSON - Read and Exporting	32
8. 2D Lists	34
a. Creating 2D Lists	34
b. Indexing 2D Lists	34
c. Double Loop 2D Lists	35
9. Dictionary Methods	35
a. Get Methods	35
b. Pop, PopItem, Clear Method	36
c. Items, Keys, Values	37
d. Update	38
e. Assignment and Copy Method	39
f. Default Dict	40
g. Double Dictionary	41
h. From Keys Method	41
10. Set	42
a. Definition and Declaration	42

b. Check Values in Sets	43
c. Union, Intersection, Difference of Set, dan Symmetric Difference	44
d. Subset dan Superset	45
<b>~ End of Beginning ~</b>	<b>46</b>
<b>[ Intermediate ]</b>	<b>47</b>
1. Function	47
a. Call Function	47
b. Arguments dan Parameters	47
c. Arbitrary Arguments	48
d. Default Parameters	48
e. Keyword Parameters	49
f. Return Statement	50
2. Apa itu Recursive Function	50
3. Lambda	51
a. Expression and Syntax	51
b. Lambda Filter	52
c. Lambda Filter	52
d. Lambda Reduce	53
4. Closure	54
a. Nested Function Concept	54
b. Non-local Variable - Local Variable vs Global Variable	55
c. Closure - Concept	55
5. Class	56
a. Definition	56
b. Instances vs Class	57
c. Declaring and Self Parameters	57
d. Constructor <code>__init__</code> Method	58
e. Instance vs Class Variables	58
f. Inheritance	61
g. Polymorphism	61
h. Encapsulation	62
i. Abstraction	65
6. Parallelism	66

a. Apa itu Concurrency dan Parallelism	66
b. Threading	67
c. Multiprocessing	68

**Akhir Kata,**

**69**



## **Intro**

---

 This note is made to remember the logic of a syntax, method, and how to use it. If this is your first time learning Python, it is advisable for you to look for other learning sources, such as YouTube, bootcamps and others.

so, let the game begins! 



Regards,  
Rakuu.

# [ Beginning ]

---

## 1. Tipe Data dan Operator

### a. Tipe Data Basic

## Tipe Data Basics

Number	Boolean	String	List	Tuple	Dictionary
int ex : 10, 200	ex : True, False	ex : 'hello' '100'	ex : [10, 20, 30]	ex : (10, 20, 30)	ex : {'name': 'David', 'age': 23, 'height': 56.5}
float ex : 3.14					
complex ex : 4 + 3j					

## Operators Basics

Beberapa **operasi aritmatika** yang bisa digunakan dalam program Python

Operator	Meaning	Action
+	Addition	Add the left operand and the right operand.
-	Subtraction	Subtract the right operand from the left operand.
*	Multiplication	Multiply the right operand from the left operand.
/	Division	Divide the left operand into the right operand. Python's division basically returns the real number value.
//	Floor Division(quotient)	Unlike /, the result of division is used to discard the decimal point or less and obtain only the integer part.
%	Remainder	It is read as a modulo operator and has nothing to do with the percentage that means ratio. Finds the remainder of the division.
**	Power	Multiply the left operand to the right operand. In the case of **0.5, a square root operation is performed.

# Susunan Prioritas Operator



Operator	Explanation
()	parenthesis operator
**	power operator
~, +, -	monadic operator
*, /, %, //	multiplication, division, remainder operator
+, -	addition, subtraction
>>, <<	bitwise movement operator
&	bitwise AND operator
^,	bitwise XOR operator, bit OR operator
<=, <, >, >=	comparison operator
==, !=	equal operator
=, %=, /=, //=, -=, +=, *=, **=	assignment operator, complex assignment operator
is, is not	identity operator
in, not in	membership test operator
not, or, and	logical operator

```
a = 10  
print(type(a))  
  
# <class 'int'>
```

Menggunakan method type() untuk mengecek tipe data.

## b. Tipe Data String

```
1 'I' + 'Love' + 'Python!' # String dapat ditambahkan ke string.  
1 'Python' * 10 # Menghasilkan string 'Python' diulang sebanyak 10 kali  
1 '*' * 50 # Menghasilkan string '*' diulang sebanyak 50 kali  
1 str(100) * 10 # String '100' diulang sebanyak 10 kali
```

- String Indexing:



Negatif: "hello[-1]" #'o'

- String Built in Func

- `split()` -> Digunakan untuk memisahkan *string* sesuai dengan *separator* yang kita inginkan dan mengembalikan hasilnya sebagai sebuah list.

```
teks = "halo, nama saya, budi"  
x = teks.split(", ")  
print(x)  
  
#output  
['halo', 'nama saya', 'budi']
```

- `islower()` -> Digunakan untuk mengecek apakah semua `element` dalam `string` adalah huruf kecil, akan mengembalikan `True` jika iya, dan `False` jika tidak

```
a = "HaLo"
b = "halo"

print(a.islower())
print(b.islower())
```

```
#output
False
True
```

- `count()`-> Digunakan untuk menghitung berapa kali sebuah `value` muncul dalam sebuah `string`

```
teks = "Halo semuanya, disini saya bersama dengan budi semuanya"

x = teks.count("semuanya")

print(x)

#output 2
```

## c. Operator Basic



```
1 100 + 20 # Temukan penambahan nilai numerik 100 dan 20
120
1 100 * 20 # Temukan perkalian nilai numerik 100 dan 20
2000
1 100 - 20 # Temukan pengurangan nilai numerik 100 dan 20
80
1 100 / 20 # Bagi nilai numerik 100 dan 20
5.0
1 100 // 20 # Bagilah nilai numerik 100 dengan 20 dan temukan hasil bagi
5
1 100 % 20 # Bagilah nilai numerik 100 dengan 20 dan temukan sisa bagi
0
```

## d. Prioritas Operator

```
1  10 + 20 * 30    # Perkalian dihitung terlebih dahulu.  
610  
1  (10 + 2) * 30   # Operator dalam kurung dihitung terlebih dahulu.  
900
```

## e. Built in Function

- Tipe data Integer
- int() -> Digunakan untuk mengubah nilai yang ditentukan menjadi bilangan bulat. Contohnya:

```
x = int("12")
```

```
y = int(12.5)
```

```
print(x)
```

```
print(y)
```

```
#output
```

```
12
```

```
12
```

- Tipe data Float
- pow() -> Digunakan untuk mendapatkan nilai pangkat dari suatu bilangan. Contohnya:

```
x = pow(3, 3) # 3 pangkat 3
```

```
y = pow(3, -3) # 3 pangkat -3
```

```
print(x)
```

```
print(y)
```

```
#output
```

```
27
```

```
0.015625
```

## 2. Variable dan Input

### a. Deklarasi Variable

Aturan-aturan dalam pembuatan variabel:

- *Identifier* dapat berupa kombinasi huruf atau angka atau garis bawah (\_), tetapi tidak dapat terdiri dari simbol khusus apa pun.
- *Identifier* tidak dapat dimulai dengan angka.
- *Identifier* tidak boleh berisi spasi atau tab.
- *Identifier* peka terhadap huruf besar dan kecil. Oleh karena itu, variabel index dan INDEX adalah variabel yang berbeda.
- *Identifier* tidak dapat menggunakan salah satu dari \*Python Reserved Words \*.

### b. Variable Assignment

```
x, y = 100, 200  
result = x + y  
print(result)
```

```
#output  
300  
num1 = num2 = num3 = 200  
print(num1, num2, num3)
```

```
#output  
200 200 200
```

### c. Compound Assignment Operators

Operator	Description	Example
<code>+ =</code>	Kombinasi dari Addition Operator dan Simple Assignment Operator	<code>i+=10</code>
<code>- =</code>	Kombinasi dari Subtraction Operator dan Simple Assignment Operator	<code>i-=10</code>
<code>* =</code>	Kombinasi dari Multiplication Operator dan Simple Assignment Operator	<code>i*=10</code>
<code>/ =</code>	Kombinasi dari Division Operator dan Simple Assignment Operator	<code>i/=10</code>
<code>^ =</code>	Kombinasi dari Bitwise XOR(^) Operator dan Simple Assignment Operator	<code>i^=10</code>
<code>% =</code>	Kombinasi dari Modulo Operator dan Simple Assignment Operator	<code>i%=10</code>

- Contoh dari Compound Assignment Operators :

```
num = 200
num += 100
print(num)
num -= 100
print(num)
num *= 20
print(num)
num /= 2
print(num)
```

```
#output
300
200
4000
2000.0
```



### d. Operator Aritmatika

Operator	Sign	Example	Value Result
Addition	+	<code>7 + 4</code>	11
Subtraction	-	<code>7 - 4</code>	3
Multiplication	*	<code>7 * 4</code>	28
Real Division	/	<code>7 / 4</code>	1.75
Integer Division (Quotiente)	//	<code>7 // 4</code>	1
Modulo	%	<code>7 % 4</code>	3
Exponentiation	**	<code>7 ** 2</code>	49

- Contoh dari Operator Aritmatika :

```
berat = float(input("Masukkan berat anda dalam kg: "))
tinggi = float(input("Masukkan tinggi anda dalam m: "))

bmi = (berat/(tinggi ** 2))
print("BMI Anda =", bmi)

#output
Masukkan berat anda dalam kg: 70
Masukkan tinggi anda dalam m: 1.75
BMI Anda = 22.857142857142858
```

## e. Input Function

- Contoh dari Input Function :

```
x = int(input("Masukkan integer pertama: "))
y = int(input("Masukkan integer kedua: "))
s = x + y
print("Total dari", x, "dan", y, "adalah", s)

#output
Masukkan integer pertama:1
Masukkan integer kedua:2
Total dari 1 dan 2 adalah 3
```

### 3. Logical and Comparison Operators

#### a. Comparison Operators

- Contoh dari Comparison Operators :

```
print('Result of 10 > 5: ', 10 > 5)
print('Result of 5 < 1: ', 5 < 1)
print('Result of 5 == 5', 5 == 5)
print('Result of 5 != 5', 5 != 5)
print("Result of 'a' > 'b': ", 'a' > 'b')
```

```
#output
Result of 10 > 5: True
Result of 5 < 1: False
Result of 5 == 5: True
Result of 5 != 5: False
Result of 'a' > 'b': False
```



#### b. Comparison Operator Is & In

- Contoh dari Operator Is & In :

```
a = 1
b = 1.0
print(a == b)
print(a is b)
```

```
# output
True
False
```

```
print('aaa' in 'aaa-bbb-ccc')
print('bbb' in 'aaa-bbb-ccc')
print('ddd' in 'aaa-bbb-ccc')
```

```
#output
True
True
False
```

### c. Logical Operators

Operator	Description
x and y	Jika salah satu dari x atau y Salah, maka Salah. Itu Benar hanya jika semuanya Benar.
x or y	Jika salah satu dari x atau y Benar, maka Benar. Itu Salah hanya jika semuanya Salah.
not x	Jika x Benar, itu Salah. Jika x Salah, maka Benar.

- Contoh dari Logical Operators :

```
x = True
y = False
print(x and y)
print(x or y)
print(not x)
print(not y)
```

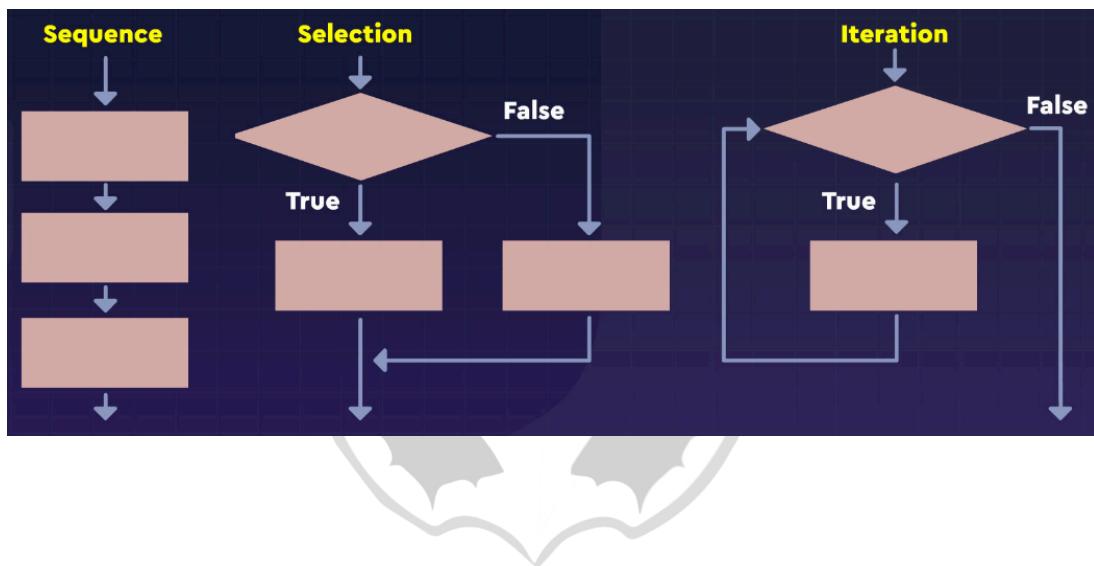
```
#output
False
True
False
True
```

## 4. Conditional Statement

### a. Selection Structure Concept

terdapat 3 tipe *control structures* dalam bahasa pemrograman:

1. **Sequence** - struktur dimana perintah dieksekusi secara berurutan.
2. **Selection** - struktur dimana salah satu dari beberapa instruksi dipilih dan dieksekusi.
3. **Iteration** - struktur dimana perintah yang sama dieksekusi berulang kali.



### b. If Statement Syntax

```
age = 18 # diberikan kondisi bahwa age = 18

if age < 20: # result dari age < 20 adalah True
    print('youth discount')

#output
youth discount
```

### c. Indentation

Indentasi dalam Python digunakan untuk menunjukkan struktur blok kode, seperti di dalam loop, fungsi, atau kondisi. Python mengandalkan indentasi untuk membedakan pernyataan yang termasuk dalam blok yang sama, sehingga indentasi yang tepat sangat penting untuk memastikan kode berjalan dengan benar.

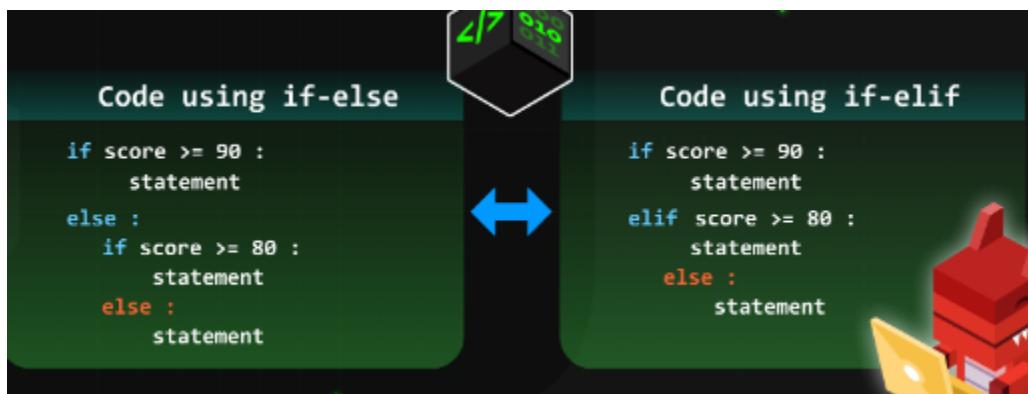
#### d. Pass Keyword

`pass` digunakan ketika Anda ingin menulis blok kode seperti *conditional*, *function*, atau *looping* pada lain waktu karena eksekusi yang tepat dari blok kode tersebut belum dapat ditentukan. Contoh:

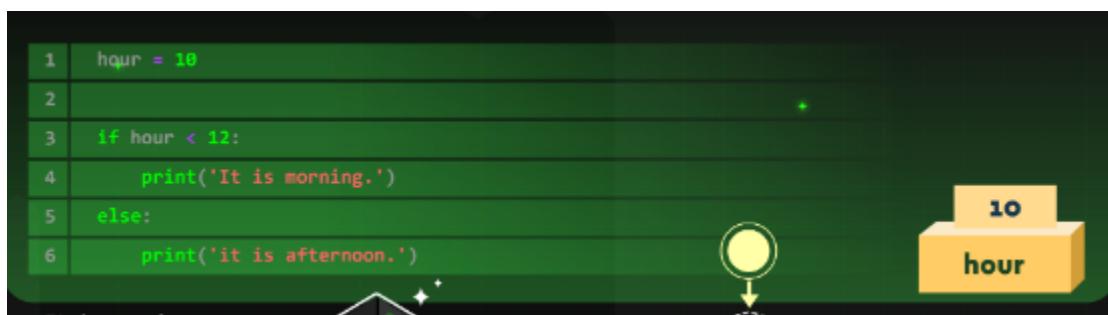
```
num = 2
if num % 2 == 0:
    print('Nomor genap')
if num % 3 == 0:
    pass
```

```
#output
Nomor genap
```

#### e. Elif Statement



#### f. If-Else Statement



## **g. Nested Conditional Statement**

- Contoh dari Nested Conditional Statement :

```
num = -100
if num < 0:
    print(num, 'adalah bilangan negatif.')
else:
    print(num, 'adalah bukan bilangan negatif')
    if num % 2 == 0:
        print(num, 'adalah bilangan genap')
    else:
        print(num, 'adalah bilangan ganjil')

#output
-100 adalah bilangan negatif
```

## **5. Loop**



### **a. Basic Loop – For Loop Syntax**

- Contoh dari For Loop :

```
for i in range(5):
    print('Welcome to everyone!!')

#output
Welcome to everyone!!
```

## b. Sequence Object Iteration

- Contoh dari Object Iteration :

```
numbers = [11, 22, 33, 44, 55, 66]
for n in numbers:
    print('n =', n)

#output
n = 11
n = 22
n = 33
n = 44
n = 55
n = 66
```

## c. While Loop Syntax

- Contoh dari Object Iteration :



```
i = 0 # initial value
while i < 5: # kode didalam block akan dieksekusi selama kondisi True
    print('Wlecome to everyone!!!')
    i += 1
```

## d. Break vs Continue Keywords

- Contoh dari Break Keywords :

```
st = 'Programming'
# fungsi akan di eksekusi selama huruf adalah konsonan
for ch in st:
    if ch in ['a','e','i','o','u']:
        break # stop loop jika menemukan huruf vokal
    print(ch)
print('The end')
```

```
#output  
P  
r  
The end
```

- Contoh dari Continue Keywords :

```
st = 'Programming'  
# fungsi akan di eksekusi selama huruf adalah konsonan  
for ch in st:  
    if ch in ['a', 'e', 'i', 'o', 'u']:  
        continue # skip eksekusi kode dibawah jika huruf vokal  
    print(ch)  
print('The end')
```

```
#output  
P  
r  
g  
r  
m  
m  
n  
g  
The end
```



## e. Nested Loop

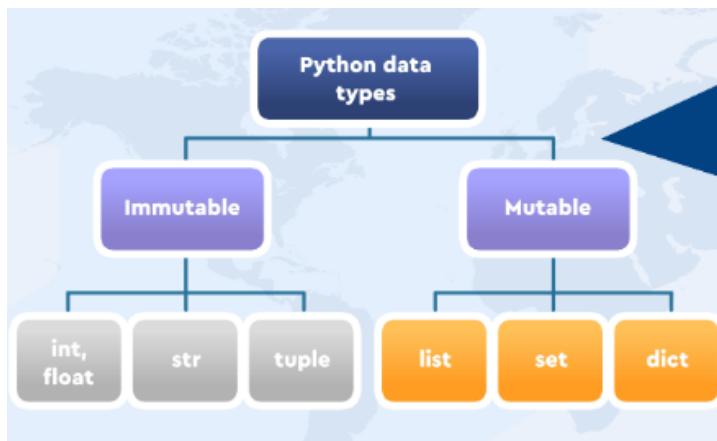
```
1  for i in range(2, 10):      #outer for Logo  
2      for j in range(1, 10): #inner for Logo  
3          print('{0}{1} = {2:2d}, '.format(i, j, i=j) end = ' ')  
4  print()                  # executes inner Loop and change line
```

2x1= 2, 2x2= 4, 2x3= 6, 2x4= 8, 2x5= 10, 2x6= 12, 2x7= 14, 2x8= 16, 2x9= 18,  
3x1= 3, 3x2= 6, 3x3= 9, 3x4= 12, 3x5= 15, 3x6= 18, 3x7= 21, 3x8= 24, 3x9= 27,  
4x1= 4, 4x2= 8, 4x3= 12, 4x4= 16, 4x5= 20, 4x6= 24, 4x7= 28, 4x8= 32, 4x9= 36,  
5x1= 5, 5x2= 10, 5x3= 15, 5x4= 20, 5x5= 25, 5x6= 30, 5x7= 35, 5x8= 40, 5x9= 45,  
6x1= 6, 6x2= 12, 6x3= 18, 6x4= 24, 6x5= 30, 6x6= 36, 6x7= 42, 6x8= 48, 6x9= 54,  
7x1= 7, 7x2= 14, 7x3= 21, 7x4= 28, 7x5= 35, 7x6= 42, 7x7= 49, 7x8= 56, 7x9= 63,  
8x1= 8, 8x2= 16, 8x3= 24, 8x4= 32, 8x5= 40, 8x6= 48, 8x7= 56, 8x8= 64, 8x9= 72,  
9x1= 9, 9x2= 18, 9x3= 27, 9x4= 36, 9x5= 45, 9x6= 54, 9x7= 63, 9x8= 72, 9x9= 81,

## 6. Sequence Data Types

### a. Mutable vs Immutable Data Types

- Jika setelah data dibuat, *value* tidak dapat diubah, maka tipe data tersebut termasuk dalam **immutable data types**. Sebaliknya, jika setelah objek tersebut dibuat dan data masih bisa diubah, maka tipe data tersebut termasuk dalam **mutable data types**.



*list* merupakan tipe data *mutable* sehingga kita dapat merubah variabel yang merupakan sebuah *list*, hal ini dapat dilakukan salah satunya menggunakan fungsi *append()*:

```
a = ['apple', 'banana', 'orange']
print(id(a))

a.append('grapes')
print(id(a))
print(a)

output:
140372445629448
140372445629448
['apple', 'banana', 'orange', 'grapes']
```

*string* merupakan tipe data *immutable* dan *value* tipe data *string* tidak dapat kita ubah selain dengan membuat objek baru.

```
message = "Welcome to Skilvul"
message[0] = 'p'
print(message)

output:
message[0] = 'p'
TypeError: 'str' object does not support item assignment
```

## b. List - Declaration



```
1 fruits = ['banana', 'apple', 'orange', 'kiwi'] # List with strings
2 print(fruits)
3 mixed_list = [100, 200, 'apple', 400]
4 print(mixed_list)
['banana', 'apple', 'orange', 'kiwi']
[100, 200, 'apple', 400]
```

Lists dalam python dapat memiliki tipe data yang berbeda

## c. List - Indexing

- Dalam sebuah *list*, indeks dari item pertama adalah 0, dan item terakhir adalah -1. Contoh kode untuk mengakses sebuah *list*:

```
n_list = [11, 22, 33, 44, 55, 66]

print(n_list[0]) # indeks item pertama dari list adalah 0

print(n_list[1]) # index item kedua dari list adalah 1

print(n_list[-1]) # index item -1 dari list adalah 66

#output
11
22
66
```

- Gunakan fungsi `len()` untuk menghitung jumlah item dalam `list`, yang dengan kata lain panjang dari sebuah `list`.

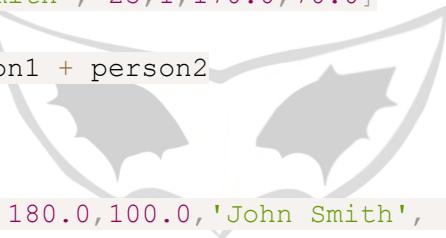
```
n_list = [11, 22, 33, 44, 55, 66]
print(n_list)
print(len(n_list))

#output
[11, 22, 33, 44, 55, 66]
```

#### d. List - Addition and Deletion

- Contoh dari Addition :

##### **+ Operator**



```
person1 = ['David Doe', 20, 1, 180.0, 100.0]
person2 = ['John Smith', 25, 1, 170.0, 70.0]

person_list = person1 + person2
print(person_list)

#output
['David Doe', 20, 1, 180.0, 100.0, 'John Smith', 25, 1, 170.0, 70.0]
```

##### **append()**

```
a_list = ['a', 'b', 'c', 'd', 'e']
a_list.append('f') # menambahkan karakter f
print(a_list)

#output
['a', 'b', 'c', 'd', 'e', 'f']
```

##### **extend()**

```
list1 = ['a', 'b', 'c']
list2 = [1, 2, 3]
list1.extend(list2)
print(list1)

list1.extend('d')
print(list1)
```

```
#output
['a', 'b', 'c', 1, 2, 3]
['a', 'b', 'c', 1, 2, 3, 'd']
```

- Contoh dari Deletion :

```
del
n_list = [11, 22, 33, 44, 55, 66]
print(n_list) # print seluruh items

del n_list[3] # delete 44
print(n_list)
```

```
#output
[11, 22, 33, 44, 55, 66]
[11, 22, 33, 55, 66]
```

### pop()

`pop()` method berfungsi kurang lebih mirip dengan `del` namun `pop()` juga mengembalikan *value index* yang dihapus. Apabila kita tidak mengisi *index* dalam metode `pop()` maka elemen **terakhir** yang akan **dihilangkan dari list**.

```
n_list = [10, 20, 30]
print(n_list) # print seluruh items
```

```
n = n_list.pop()
print('n = ', n)
print('n_list = ', n_list)
```

```
#output
[10, 20, 30]
n = 30
n_list = [10, 20]
```

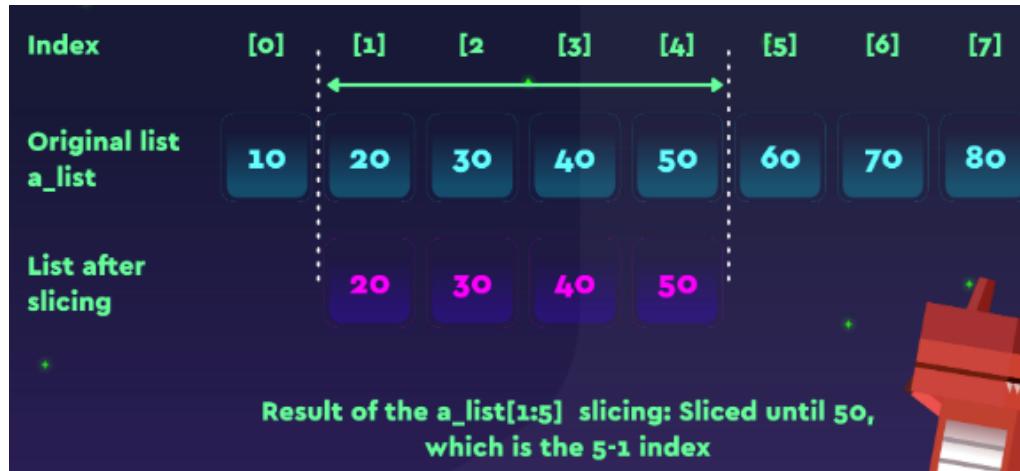
## **remove()**

```
n_list = [11, 22, 33, 44, 55, 66]  
print(n_list)
```

```
n_list.remove(44)  
print(n_list)
```

```
#output  
[11, 22, 33, 44, 55, 66]  
[11, 22, 33, 55, 66]
```

## e. List - Slicing



```
1 a_lists = (10, 20, 30, 40, 50, 60, 70, 80)  
2 a_list[1:5]  
[20, 30, 40, 50]  
1 a_list[0:5]  
[10, 20, 30, 40, 50]  
1 a_list[1:]  
[20, 30, 40, 50, 60, 70, 80]  
1 a_list[:5]  
[10, 20, 30, 40, 50]  
1 a_list[:]  
[10, 20, 30, 40, 50, 60, 70, 80]
```

Syntax	Function
a_list[start:end]	Slicing item dari awal hingga (akhir-1) (tidak termasuk item dari indeks akhir).
a_list[start:]	Slicing dari awal hingga akhir list. Slicing bagian akhir dari list.
a_list[:end]	Slicing dari awal hingga akhir -1 indeks.
a_list[:]	Slicing keseluruhan list.
a_list[start:end:step]	Slicing dengan melewatkkan satu step dari awal hingga akhir -1.
a_list[-2:]	Slicing dua item dari akhir.
a_list[:-2]	Slicing seluruh item dari awal kecuali dua item terakhir.
a_list[::-1]	Import semua item tetapi dalam urutan terbalik.
a_list[1::-1]	Slicing dua item pertama dalam urutan terbalik.

## f. List - In Operator

```

1   a_lists = (10, 20, 30, 40)
2   10 in a_list
True ←
1   50 in a_list
False
1   10 not in a_list
False
1   50 not in a_list
True
1   for n in a_list:
2   print(n, end=' ')
10 20 30 40

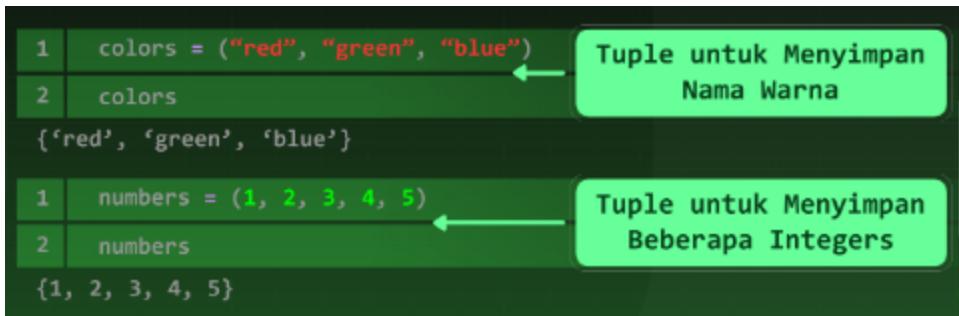
```

Sebelum melanjutkan dengan metode a\_list.remove(10), kita dapat memeriksa apakah item 10 ada dengan operasi "In" dan apakah itu mengembalikan True, sehingga membuat metode remove() tidak akan menghasilkan error.



## g. Tuple - Basic

*Tuple* adalah sebuah *immutable list*, dimana setelah dibuat, *tuple* tidak dapat diubah dengan cara apa pun. *Tuple* didefinisikan mirip dengan sebuah *list*, perbedannya adalah *list* menggunakan **tanda kurung siku** [ ] namun *tuple* menggunakan **tanda kurung** () .



## h. Sequences Data Types - Operator

Tipe data sequence dapat menggunakan operator perkalian, yang membuat elemen objek berulang kali. Namun, perkalian objek tidak mungkin dilakukan untuk *range* dengan operator `*`. Untuk melakukan *iteration*, kita dapat menulisnya sebagai [tipe data urutan] \* integer.

Contoh-contoh *iteration* dalam tipe data sequence

- *List*

```

list1 = [11, 22, 33, 44] * 2
print(list1)

#output
[11, 22, 33, 44, 11, 22, 33, 44]

```

- *Tuples*

```

tup1 = (1, 2, 3)
print(tup1 * 2)

#output
(1, 2, 3, 1, 2, 3)

```

- *String*

```

str2 = 'hello'
print(str2 * 3)

#output
hellohellohello

```

- Range (harus dijadikan list/tuple dulu supaya bisa)

```
ran = list(range(5)) * 3
```

```
print(ran)
```

```
#output
```

```
[0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4]
```

```
ran = tuple(range(5)) * 3
```

```
print(ran)
```

```
#output
```

```
(0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4)
```



### i. Sequences Data Types - Count Method

Method `count()` mengembalikan jumlah elemen dalam objek `sequence`. Metode ini bisa digunakan untuk **mencari elemen** yang terdapat di dalam objek `sequence` dan juga mengetahui **berapa banyak elemen** di dalam objek `sequence` tersebut.

```
list1 = [11, 11, 11, 22, 33, 44]
print(list1.count(11)) # mencari total angka 11 dalam list
```

```
#output
```

```
3
```

## j. Sequences Data Types - Operators

- Contoh dari Operator in dan not in :

```
list1 = [10, 20, 30, 40]
print(10 in list1)
print(10 not in list1)
```

```
#output
True
False
```

## 7. Dictionary



### a. Basic

- Contoh basic dari Dictionary :

```
person = {'Name' : 'David Doe', 'Age' : 26, 'Weight' : 82} #  
Dictionary dengan nama, umur dan berat
```

## b. Key - Value Pair

Item	Key	Value
Item 1	'Name'	'David Doe'
Item 2	'Age'	26
Item 3	'Weight'	82

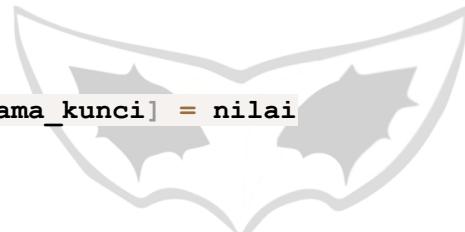
```
1 person = {'Name' : 'David Doe', 'Age' : 26, 'Weight' : 82}
1 person ['Name']
David Doe
1 person ['Age']
26
1 person ['Weight']
82
```



## c. Adding, Modifying, Delete Values

### Adding Items:

```
nama_dictionary[nama_kunci] = nilai
```



### Contoh:

```
person = {'Name': 'David Doe', 'Age': 26, 'Weight' : 82}
person['Job'] = 'Data Scientist' # Key baru: masukkan nilai dari key tersebut
print(person)
#output
{'Name': 'David Doe', 'Age': 26, 'Weight' : 82, 'Job': 'Data
Scientist'}
```

### Modifying Items:

```
person = {'Name': 'David Doe', 'Age': 26, 'Weight' : 82}
person['Age'] = 27 # Ubah value dari key yang sudah ada 'Age'
print(person)
#output
{'Name': 'David Doe', 'Age': 27, 'Weight' : 82}
```

## Deleting Items:

```
person = { 'Name': 'David Doe', 'Age': 26, 'Weight' : 82}
del person['Age'] # Delete value dari key yang sudah ada 'Age'
print(person)
#output
{ 'Name': 'David Doe', 'Weight' : 82}
```

## d. Function and Operators

### len() function dan in operator

Untuk mengetahui jumlah *item* dalam sebuah *dictionary*, kita bisa menggunakan fungsi `len()` seperti berikut:

```
a = {'abc':100, 'def':200}
print(len(a))
```

```
#output
2
```

`in` operator bisa digunakan untuk **memvalidasi** apakah sebuah *key* ada di dalam sebuah *dictionary*. Operator `in` akan mengembalikan nilai *True* apabila *key* tersebut ditemukan dalam *dictionary* dan operator `not in` melakukan kebalikan dari operator `in`.

```
1 * person = { 'Name' : 'David Doe', 'Age' : 26, 'Weight' : 82}
1 len(person) # Mengembalikan jumlah item dalam dictionary
3
1 'Name' in person # 'Nama' ditemukan sebagai salah satu kunci
True
1 'Job' in person # 'Job' tidak ditemukan sebagai salah satu kunci
False
1 'David Doe' in person # 'David Doe' ditemukan di
    nilai tetapi tidak di kunci (hati-hati)
False
1 'David Doe' not in person # Mengembalikan True karena
    'David Doe' tidak ditemukan di kunci
True
```

## e. JSON Datatypes - Basics

Berikut adalah contoh bentuk data dalam JSON :

```
{  
    "Name": "David Doe", → kunci: String digunakan sebagai kunci dan nilai  
    "Age": 25, → Menggunakan integer sebagai nilai  
    "Hobby": ["basketball"] → List juga bisa digunakan sebagai nilai  
    "Family": {"father": "John Doe", "mother": "Marry Doe"}, → Dictionary sebagai nilai  
    "Married": true → Serta Boolean sebagai nilai  
}
```

## f. JSON - Read and Exporting

jadi, fungsi .loads() dalam python itu untuk mengubah json menjadi dictionary. nahh sedangkan .load() berfungsi untuk mengimpor file JSON untuk digunakan ke python.

lalu, fungsi .dumps() dalam python itu untuk mengubah dictionary menjadi JSON. nahh sedangkan .dump () berfungsi untuk mengekspor file JSON

### JSON TO DICTIONARY:

```
1 import json  
2  
3 data = '{"Name": "David Doe", "Age": 25, "Hobby": ["basketball"],\n4 "Family": {"father": "John Doe", "mother": "Marry Doe"}, "Married": true}'  
5  
6 json_data = json.loads(data)  
7  
8 print(type(json_data))  
9 print(json_data['Name'])  
10 print(json_data['Family'])  
11 print(json_data['Married'])  
  
<class 'dict'>  
David Doe  
{'father': 'John Doe', 'mother': 'Marry Doe'}  
True
```

Sementara itu kebalikannya, kita bisa menggunakan fungsi `dumps()` untuk merubah *dictionary* menjadi *json-string*. Fungsi `dumps()` untuk menghasilkan sebuah file tipe *JSON* dari data *dictionary*.

```
import json

Dictionary = {'Halo':123, 'Semua':456}

json_string = json.dumps(Dictionary)

print('Result: ', json_string)
print('Tipe: ', type(json_string))

# OUTPUT
Result: {"Halo": 123, "Semua": 456}
Tipe: <class 'str'>
```

Result `json_string` merupakan sebuah *JSON string* hasil dari fungsi `json.dumps()`



### Exporting JSON data

Untuk membuat format data *JSON* ke dalam sebuah *file* pada Python, kita dapat menggunakan `json.dump()`.

```
1 import json
2
3 data = '{"title": "The Old man and The Sea", "ISBN": "12345", "Author": "En
4 json_data = json.loads(data)
5
6 #Code to create json data as book.json file
7 with open('book.json', 'w') as f:
8     json.dump(json_data, f, indent='\t')
```

```
8. Dictionary Data Types > JSON > data.json > ...
1  {
2      "name": "Rakha",
3      "age": 17,
4      "is_student": true,
5      "is_employed": false,
6      "address": null,
7      "hobbies": [
8          "coding",
9          "designing",
10         "traveling"
11     ],
12     "grades": {
13         "math": 95,
14         "english": 88
15     }
16 }
```

# contoh file .json yang dihasilkan melalui .dump().

## 8. 2D Lists



### a. Creating 2D Lists

```
list_array = [[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]]
```

### b. Indexing 2D Lists

```
list_array = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
list_array[0]

#output
[1, 2, 3]
```

```
list_array = [[1,2,3],[4,5,6],[7,8,9]]  
list_array[0][2]
```

```
#output  
3
```

### c. Double Loop 2D Lists

Karena ini adalah sebuah *2D list*, maka tipe data dari masing-masing elemen tersebut adalah **sebuah list**.

```
list_array = [[1,2,3],[4,5,6],[7,8,9]]  
for item in list_array:  
    print('item =',item)
```

```
#output  
item = [1,2,3]  
item = [4,5,6]  
item = [7,8,9]
```



## 9. Dictionary Methods

### a. Get Methods

Fungsi `get()` dapat kita gunakan untuk mendapatkan value dari key yang spesifik.

```
dic = {'a':10,'b':20,'c':30,'d':40}  
a = dic.get('a') # fungsi get untuk mendapatkan value dari key a  
print(a)  
  
#output  
10
```

Kita juga dapat memberikan *default value* dalam sebuah fungsi `get()`.

```
#syntax  
dic.get(key, default_value)
```

Sehingga jika kita menggunakan fungsi `get()` untuk sebuah *key* yang tidak ada di dalam sebuah *dictionary* maka fungsi `get()` akan mengembalikan *default value* tersebut.

```
dic = {'a':10,'b':20,'c':30,'d':40}  
b = dic.get('z',0) # tidak terdapat key z dalam dictionary dic  
sehingga dikembalikan value 0  
print(b)  
  
#output  
0
```

## b. Pop, PopItem, Clear Method



### - `pop()`

Fungsi `pop()` dalam *dictionary* dapat digunakan sebagai salah satu metode untuk **menghapus item** dari sebuah *dictionary*. Fungsi `pop()` akan menghapus sebuah *key-value pair* yang spesifik dan **mengembalikan value yang dihapus**.

```
dic = {'a': 10, 'b': 20, 'c': 30, 'd': 40}  
print(dic.pop('a')) # meremove key-value pair dengan key a  
print(dic)  
#output  
10  
{'b': 20, 'c': 30, 'd': 40}
```

### - `popitem()`

Fungsi `popitem()` juga merupakan metode untuk **menghapus sebuah elemen** dari *dictionary*, namun fungsi `popitem()` menghapus *key-value pair* apa pun dari sebuah *dictionary* dan mengembalikan *key-value pair* yang dihapus dalam bentuk tipe data *tuple*.

```
dic = {'a': 10, 'b': 20, 'c': 30, 'd': 40}  
print(dic.popitem())
```

```
#output  
( 'd', 40)
```

#### - **clear()**

Untuk **menghapus seluruh key-value pair** dari sebuah *dictionary*, kita dapat menggunakan fungsi **clear()**.

```
dic = {'a': 10, 'b': 20, 'c': 30, 'd': 40}  
print(dic)  
dic.clear() # meremove semua value dalam dictionary dic  
print(dic)  
#output  
{ 'a': 10, 'b': 20, 'c': 30, 'd': 40}  
{ }
```



### c. Items, Keys, Values

#### - **keys()**

```
dic = {'a': 10, 'b': 20, 'c': 30, 'd': 40}  
print(dic.keys()) # mengakses seluruh key yang ada dalam dictionary  
  
#output  
dict_keys(['a', 'b', 'c', 'd'])
```

#### - **values()**

```
dic = {'a': 10, 'b': 20, 'c': 30, 'd': 40}  
print(dic.values()) # mengakses seluruh values yang ada dalam  
dictionary  
  
#output  
dict_values([10, 20, 30, 40])
```

### - **items()**

```
dic = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
print(dic.items()) # mengakses seluruh items yang ada dalam
                  dictionary

#output
dict_items([('a', 10), ('b', 20), ('c', 30), ('d', 40)])

# kita juga bisa menggunakan *for loop* untuk mengakses seluruh
*item* satu per satu

for str1,num, in dic.items():
    print(str1, ':', num)

#output
a : 10
b : 20
c : 30
d : 40
```



### d. Update

Untuk mengupdate *value* dalam *dictionary* kita bisa menggunakan fungsi **update()**. Fungsi **update(key=value)** memodifikasi nilai *key* dalam sebuah *dictionary*.

Namun jika kita menggunakan fungsi **update()** untuk sebuah *key* yang belum tersedia, maka fungsi **update()** akan **berfungsi untuk menambahkan key-value pair tersebut ke dalam *dictionary***:

```
dic = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
print(dic)

dic.update(a=900, f=60) # mengupdate value a dan menambahkan key dan
                       value f
print(dic)

#output
{'a': 10, 'b': 20, 'c': 30, 'd': 40}
{'a': 900, 'b': 20, 'c': 30, 'd': 40, 'f': 60}
```

## e. Assignment and Copy Method

### - `copy()`

Dalam sebuah *dictionary* kita bisa menggunakan fungsi `copy()` untuk membuat replika dari *dictionary* tersebut.

```
x = {'a': 0, 'b': 0, 'c': 0, 'd': 0}
y = x.copy()

print(x is y) # cek apakah kedua dictionary merupakan objek yang sama
print(x == y) # cek apakah key-value pairnya sama

#output
False
True
```

Dari contoh di atas kita sudah berhasil meng-copy *dictionary* x ke *variabel* y. Sekarang membandingkan x dan y dengan operator `is` akan mengembalikan nilai False.

Dengan kata lain, kedua *dictionary* tersebut adalah **objek yang berbeda**. Namun, *key-value pair* yang kita copy adalah sama, jadi jika kita membandingkannya dengan operator `==` akan mengembalikan nilai True.

### - `deepcopy()`

Apabila kita ingin membuat replika dari sebuah *double dictionary*, kita tidak bisa menggunakan metode `copy()`. Maka, solusinya adalah menggunakan `deepcopy()`

```
import copy
#syntax
y = copy.deepcopy(dictionary_yang_ingin_dicopy)
x = {'a': {'python':'2.7'}, 'b':{'python':'3.6'}}
import copy # memanggil copy module
y = copy.deepcopy(x) # deep copy menggunakan deepcopy function dari
copy module

y['a']['python'] = '2.7.15' # merubah value dari key a - python
print(x)
print(y)
```

```
#output
{'a': {'python': '2.7'}, 'b': {'python': '3.6'}}
{'a': {'python': '2.7.15'}, 'b': {'python': '3.6'}}
```

## f. Default Dict

Mari kita lihat kasus di mana `KeyError` muncul di tipe data `dictionary`.

| Terjadi kesalahan saat Anda mencoba mengakses key yang tidak ada di dalam `dictionary(dic)`.

```
1 x = {'a' : 0, 'b' : 0, 'c' : 0, 'd' : 0}
2 x['z']

-----
KeyError Traceback (most recent call last)
<ipython-input-99-7867b3f4c074> in <module>
      1 x = {'a' : 0, 'b' : 0, 'c' : 0, 'd' : 0}
----> 2 x['z']
KeyError: 'z'
```

Dari contoh di atas kita bisa melihat bahwa jika kita **mencoba mengakses key z** dari sebuah **dictionary x** yang tidak memiliki **key tersebut**, maka akan muncul **KeyError**.

Sehingga untuk mencegah hal ini terjadi kita butuh fungsi `defaultdict()`. Fungsi `defaultdict()` ini akan membantu kita agar mencegah `KeyError` dan akan mengembalikan `default value`.

```
from collections import defaultdict # import default dict method dari
collections module

keys = ('a', 'b', 'c', 'd')
y = dict.fromkeys(keys, 100)
y = defaultdict(int) # set default value sebagai integer

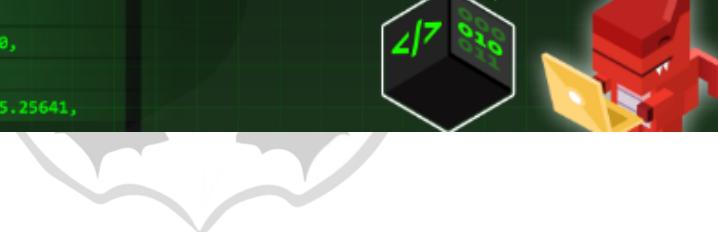
print(y['z']) # mengakses key z dari dictionary y

#output
0
```

Pada contoh di atas kita mencoba mengakses *key* z yang tidak ada di dalam *dictionary* y, namun karena kita sudah memanggil fungsi *defaultdict()* dan kita sudah set *default value* sebagai *int* dan *default value* dari *int* adalah 0, maka program akan mengembalikan nilai 0 dan *KeyError* tidak terjadi.

## g. Double Dictionary

```
1 terrestrial_planet = {  
2     'Mercury': {  
3         'mean_radius': 2439.7,  
4         'mass': 3.3022E+23,  
5         'orbital_period': 87.969  
6     },  
7     'Venus': {  
8         'mean_radius': 6051.8,  
9         'mass': 4.8676E+24,  
10        'orbital_period': 224.70069,  
11    },  
12    'Earth': {  
13        'mean_radius': 6371.0,  
14        'mass': 5.97219E+24,  
15        'orbital_period': 365.25641,  
16    },  
17    'Mars': {  
18        'mean_radius': 3389.5,  
19        'mass': 6.4185E+23,  
20        'orbital_period': 686.9600,  
21    }  
22 }  
23  
24 print(terrestrial_planet['Venus']['mean_radius'])  
6051.8
```



## h. From Keys Method

Ada berbagai cara untuk membuat *dictionary* dan salah satunya adalah menggunakan *\*\*fromkeys()*

```
keys = ['a', 'b', 'c', 'd']  
x = dict.fromkeys(keys)  
print(x)  
  
#output  
{'a': None, 'b': None, 'c': None, 'd': None}
```

Kode di atas menunjukkan cara membuat sebuah *dictionary* x dengan key berdasarkan data dari sebuah *list keys*. Key yang ada di dalam *dictionary* x berasal dari *keys* dengan *value* awal yaitu *None*.

```

#syntax
dict.fromkeys(key_list,value)

keys = ('a', 'b', 'c', 'd')
y = dict.fromkeys(keys,100)
print(y)

#output
{'a':100 , 'b': 100, 'c': 100, 'd': 100}

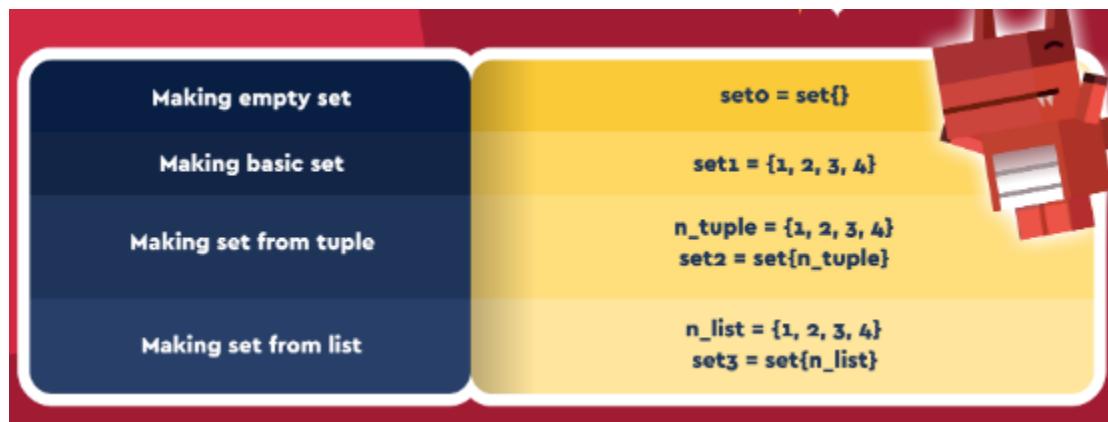
```

## 10. Set

### a. Definition and Declaration

Tipe data *set* adalah tipe data dasar yang disediakan oleh Python tetapi tidak disediakan sebagai tipe dasar dalam bahasa pemrograman lain seperti C atau Java. Ada beberapa poin penting dalam tipe data *set* :

- Dalam *set*, tidak diperbolehkan ada data yang sama atau duplikat sehingga data di dalam *set* sifatnya adalah **unik**.
- Tipe data *set* adalah *unordered* sehingga *item* dapat muncul **dalam urutan yang berbeda** setiap kali kita menggunakannya dan kita tidak bisa mengakses data menggunakan *index* atau *key* seperti tipe data *list* atau *dictionary*.
- Tipe data *set* juga **tidak bisa kita ubah** setelah kita membuatnya



## b. Check Values in Sets

Untuk mengakses **value** yang ada di dalam **set**, kita bisa menggunakan **for loop** untuk melakukan \*iterasi \*untuk mendapatkan *item* yang ada di dalam **set**

```
numbers = {2, 1, 3}
for x in numbers:
    print(x)

#output
2
1
3
```

Selain itu kita juga bisa menambahkan serta menghapus **value** yang ada di dalam **set** menggunakan fungsi **add()** serta **remove()**: Kita bisa **menambahkan** *item* baru menggunakan fungsi **add()**:

```
numbers = {1, 2, 3}
numbers.add(4)
print(numbers)

#output
{1, 2, 3, 4}
```



Dan kita bisa **menghapus item** dalam set menggunakan fungsi **remove()**:

```
numbers = {1, 2, 3, 4}
numbers.remove(4)
print(numbers)

#output
{1, 2, 3}
```

### c. Union, Intersection, Difference of Set, dan Symmetric Difference

Dalam tipe data *set*, terdapat beberapa operator yang kita bisa gunakan seperti :

- & untuk *intersection*
- | untuk *union*
- - untuk *difference of set* dan,
- ^ untuk *symmetric difference*

```
s1 = {1,2,3,4,5,6}
s2 = {4,5,6,7,8,9}

print(s1 | s2) # menggabungkan (combine)
# | untuk union (mengambil semua data yang ada dari s1 dan s2,
# bisa juga s1.union(s2))
#output {1, 2, 3, 4, 5, 6, 7, 8, 9}

print(s1 & s2) # mengambil irisan
# & untuk intersection (mengambil irisan, bisa juga
# s1.intersection(s2))
# output {4, 5, 6}
```

```

s1 = {1,2,3,4,5,6}
s2 = {4,5,6,7,8,9}

print(s1 - s2) # - untuk difference of set, berarti mengambil
data yang HANYA ADA DI S1 ATAU S2. misal s2 - s1 berarti
mengambil data yang ada di s2, dan tidak ada di s1.
#output {1, 2, 3}

print(s1 ^ s2) # ^ untuk symmetric difference, berarti
mengambil bilangan yang tidak bertabrakan, yang artinya
mengambil data yang hanya di s1 dan s2. menghiraukan yang
bertabrakan
#output {1, 2, 3, 7, 8, 9}

```



#### d. Subset dan Superset

Dalam tipe data *set*, terdapat istilah *superset* dan juga *subset* yang berarti:

- Sebuah *set* dapat dikatakan *superset* apabila **mengandung semua elemen dari set yang lain**.
- Dan sebuah *set* dapat dikatakan *subset* apabila **seluruh elemen terkandung dari set yang lain**.

```

1 s1 = {1, 2, 3, 4, 5} *
2 s2 = {1, 2, 3} *
3 s3 = {1, 2, 6}

1 s2.issubset(s1) # apakah s2 adalah subset dari s1
True
1 s3.issubset(s1) # apakah s3 adalah subset dari s1
False
1 s1.issuperset(s2) # apakah s1 adalah superset dari s2
True
1 s1.issuperset(s3) # apakah s1 adalah superset dari s3
False

```



**~ End of Beginning ~**

# [ Intermediate ]

---

## 1. Function

### a. Call Function

```
def print_star():
    print("*****")
print_star() # Memanggil function print_star (1)
print_star() # Memanggil function print_star (2)
print_star() # Memanggil function print_star (3)
print_star() # Memanggil function print_star (4)
```

```
#output
*****
*****
*****
*****
*****
```



### b. Arguments dan Parameters

```
userInput = 'rakha'

def ini_func(sapaaku): # ini param

    print('haii, salam kenal! ', sapaaku)

    return sapaaku

ini_func(userInput) #ini argument
```

### c. Arbitrary Arguments

```
def func (*sapaaku):  
    for name in sapaaku:  
        print('haloo', name)  
  
func('rakha', 'alpin', 'ahdan') # sama seperti rest parameters  
di javascript.
```

Kita bisa meneruskan 3 dan 2 argumen ke dalam **function yang sama**.

### d. Default Parameters

Jika sebuah *function* memiliki sebuah parameter yang harus dipenuhi, maka apa yang akan terjadi jika kita tidak memberikan argumen ke dalam *function* tersebut? Jika hal tersebut terjadi maka akan muncul *error*, dan untuk mengatasinya kita memerlukan *default parameters*.

Kita tetap bisa menyertakan argumen ke dalam *function* yang memiliki *default parameters* dan jika kita menyertakan argumen, maka **nilai yang diambil adalah nilai yang kita sertakan** bukan *default parameters*. *Default parameters* hanya akan diambil apabila tidak ada argumen yang kita berikan pada saat memanggil sebuah *function*.

```
def print_star(n = 1):  
    for _ in range(n):  
        print("*****")  
  
print_star(2) # memanggil print_star dengan argument n = 2  
  
#output  
*****  
*****
```

## Default Parameters in Multiple Parameters

Perlu diingat apabila kita memiliki **lebih dari 1 parameter**, *default parameters* harus ditetapkan ke semua variabel atau dari variabel terakhir dalam **urutan kemunculan parameter**. Sehingga jika kita mempunyai 3 parameter `def func(a, b, c)`, maka semuanya harus memiliki *default variable* atau:

- parameter c memiliki *default variable* dan parameter a & b tidak,
- parameter c dan b memiliki *default variable* dan parameter a tidak.

```
def add_numbers(a, b=0, c=0):  
  
    return a + b + c  
  
  
  
  
  
  
print(add_numbers(5))    # Output: 5  
  
  
  
  
  
  
print(add_numbers(5, 10))  # Output: 15  
  
  
  
  
  
  
print(add_numbers(5, 10, 15)) # Output: 30
```

## e. Keyword Parameters

```
def get_root(a,b,c):  
    r1 = (-b + (b ** 2 - 4 * a * c) ** 0.5) / (2 * a)  
    r2 = (-b - (b ** 2 - 4 * a * c) ** 0.5) / (2 * a)  
    return r1,r2  
  
result1, result2 = get_root(a=1,c=-8,b=2)  
print('Hasil akar-akarnya adalah', result1, 'atau', result2)  
  
#output  
Hasil akar-akarnya adalah 2.0 atau -4.0
```

## f. Return Statement

```
def get_sum(a,b): # fungsi yang akan mengembalikan total dari 2 angka
    result = a + b
    return result # mengembalikan hasil penjumlahan menggunakan return

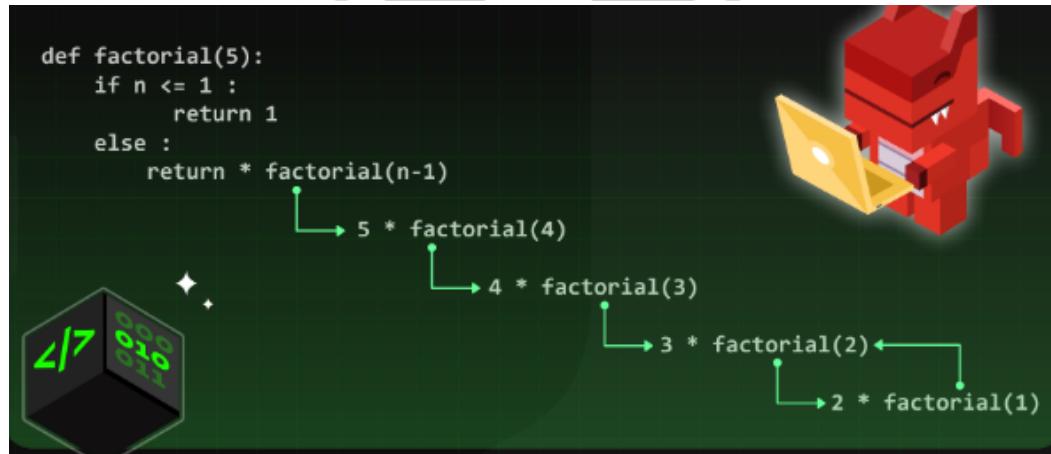
n1 = get_sum(10,20)
print("hasil penjumlahan dari 10 dan 20 adalah",n1)

#output
hasil penjumlahan dari 10 dan 20 adalah 30
```

## 2. Apa itu Recursive Function

*Recursive function* adalah *function* yang memanggil dirinya sendiri di dalam suatu *function*. Rekursi adalah **teknik pemecahan masalah** yang sangat berguna dan cukup sering dipakai dalam pembuatan program.

Contoh Recursive Function salah satunya yaitu menghitung Faktorial:



Alur fungsi rekursif faktorial adalah sebagai berikut. `faktorial(3)` mengembalikan `3 * faktorial(2)` dan `faktorial(2)`, mengembalikan `2 * faktorial(1)`. Kemudian hal ini terjadi secara berulang dan hasil pengembaliannya adalah `120`

```

recursive.py
def factorial(n): # membuat function untuk factorial
    if n <= 1: # untuk melakukan termination/pemutusan recursive
        function
        return 1
    else:
        return n * factorial(n-1) # implementasi dari recursive function

n = 5
print(f"factorial dari {n} adalah {factorial(n)}")

#output
factorial dari 5 adalah 120

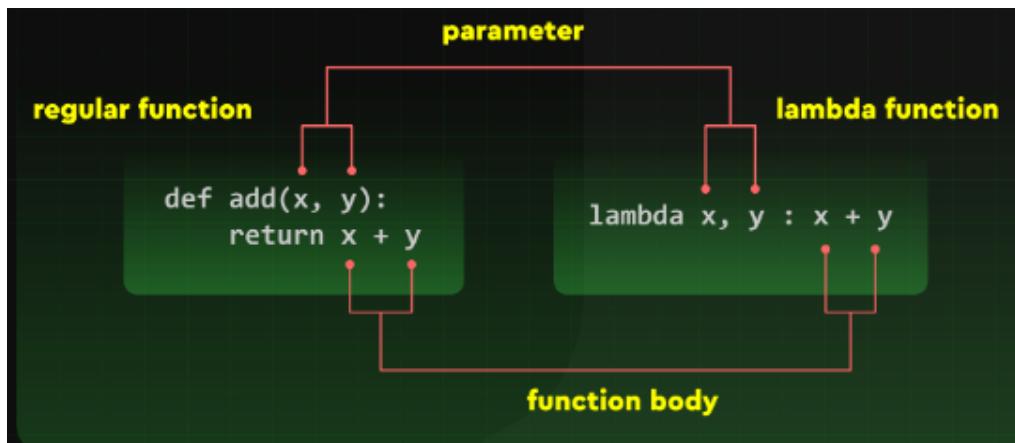
```

### 3. Lambda

#### a. Expression and Syntax

*Lambda function* adalah fungsi tanpa nama dan disebut juga *lambda expression*. Karena karakteristik seperti itu, *lambda function* juga disebut *anonymous function*. Perbandingan fungsi reguler dan fungsi lambda :

Fungsi **reguler** terdiri dari kata **kunci def, nama function, parameter, titik dua dan body function**, sedangkan fungsi **lambda** hanya memiliki parameter dan **body function**.



*Lambda function* hanya boleh memiliki satu *expression* tapi boleh memiliki beberapa parameter. Sintaks dari *lambda function* adalah sebagai berikut:

```
add = lambda x, y : x + y
print("total dari 100 dan 200 adalah:", add(100, 200))

#output
total dari 100 dan 200 adalah: 300
```

### b. Lambda Filter

*Function filter()* menerima *iterable elements* dan hanya mengembalikan elemen yang *True* dalam satu bundel. Sintaks dari *function filter()* adalah sebagai berikut:

```
filter((fungsi_yang_akan_diaplikasikan, {iterable object})
```

```
list_umur = [34, 39, 20, 18, 13, 54]

print("Umur yang dewasa: ")

for a in filter(lambda x: x >= 19, list_umur): # filter umur menggunakan
    fungsi filter

    print(a, end = ' ')

#output
Umur yang dewasa:
34 39 20 54
```

### c. Lambda Filter

*map()* mengeksekusi sebuah *function* untuk setiap *iterable items*. Setiap *item* akan dijadikan argumen pada *function* yang diinginkan. Sintaks dari *map()* yaitu sebagai berikut:

```
map(function_to_be_applied , iterable_object, ...)
```

```

a = [1, 2, 3, 4, 5, 6, 7]

a_kuadrat = list(map(lambda x: x **2, a))
print(a_kuadrat)

#output
[1, 4, 9, 16, 25, 36, 49]

```

#### d. Lambda Reduce

`reduce()` termasuk merupakan bagian dari *module* `functools`. `reduce()` mengembalikan sebuah nilai dengan menjalankan operasi dengan fungsi yang diberikan pada *item* dari sebuah *iterables object* (*list,tuple,range*). Hampir sama dengan `map()` namun `reduce()` mengembalikan nilai tunggal atau *single value*

```

from functools import reduce

a = [1,2,3]

n = reduce(lambda x,y : x + (x*y), a)

print(n)

# Langkah 1: x = 1, y = 2 → 1 + (1 * 2) = 3
# Langkah 2: x = 3 (dari iterasi pertama), y = 3 → 3 + (3 * 3)
= 12

```

## 4. Closure

### a. Nested Function Concept

Konsep *function* di dalam suatu *function* disebut *nested function*. Berbeda dengan *function* yang terletak di luar, ia dapat dengan bebas membaca variabel dari *function* induknya atau *outer function*.

```
def another_func():
    print('hello')

def outer_func():
    return another_func()

outer_func()

#output
hello
```



```
def decorate(style = 'italic'):
    def italic(s):
        return '<i>' + s + '</i>'
    def bold(s):
        return '<b>' + s + '</b>'

    if style == 'italic':
        return italic
    else:
        return bold
```

```
dec = decorate()
print(dec('hello'))
dec2 = decorate('bold')
print(dec2('hello'))
```

```
#output
<i>hello</i>
<b>hello</b>
```

## b. Non-local Variable - Local Variable vs Global Variable

### Global Keyword

Dengan bantuan `global` keyword kita bisa menjadikan *local variable* menjadi *global variable*. Berikut cara penggunaan `global` keyword di dalam kode:

```
def print_counter():
    global counter
    counter = 200
    print('counter didalam fungsi =',counter) # nilai counter didalam
    fungsi

counter = 100
print_counter()
print('counter diluar fungsi = ',counter) # nilai counter diluar
fungsi

#output
counter didalam fungsi = 200
counter diluar fungsi = 200
```



## c. Closure - Concept

Contoh Perilaku:

```
def closure_calc():
    a = 2

    def mult(x):
        return a * x

    return mult

c = closure_calc()    # Mengembalikan fungsi mult, tetapi belum
dijalankan

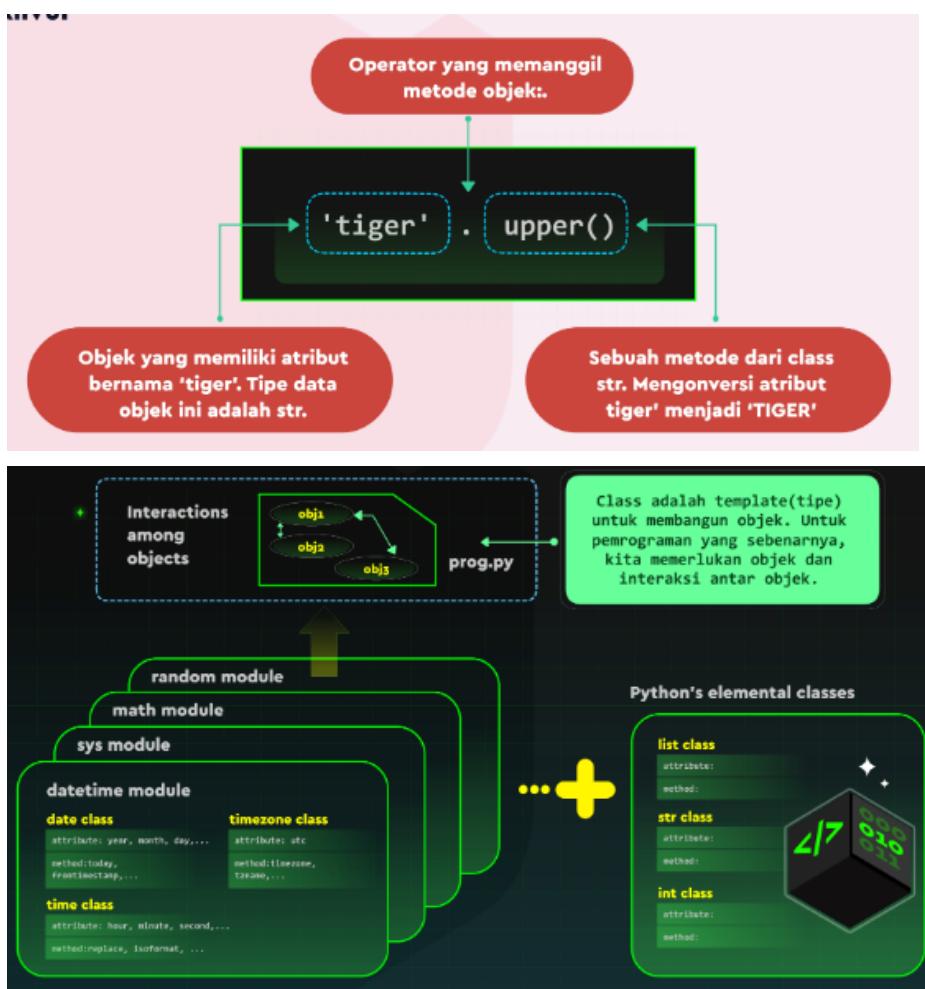
print(c(1))    # Baru di sini fungsi mult dijalankan, dengan x=1
```

Penggunaan *closure* digunakan sebagian besar untuk hal-hal berikut:

1. Membatasi penggunaan *global variables*
2. Menyembunyikan data (*data hiding*)

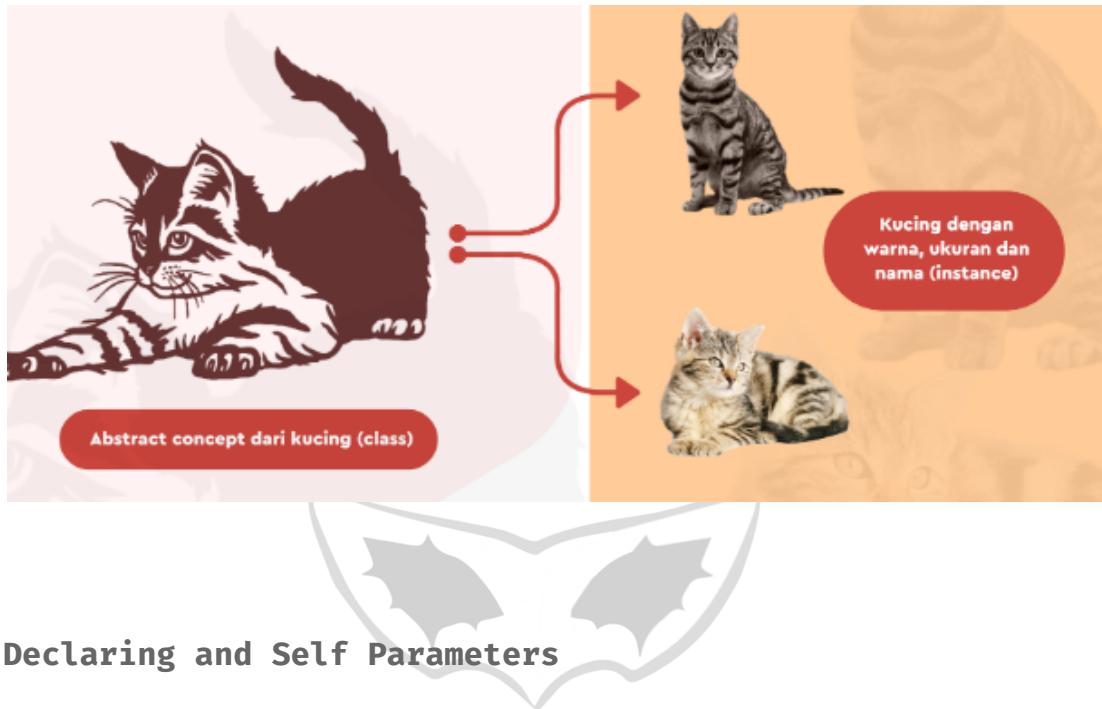
## 5. Class

### a. Definition



## b. Instances vs Class

- **Class** : Sebuah konsep abstrak yang menunjukkan **satu set atribut** dan tindakan yang digunakan dalam sebuah program.
- **Instances** : Sebuah objek individu yang dibuat dari *class*. Instances memiliki **nilai atribut tertentu yang spesifik**.



## c. Declaring and Self Parameters

```
class Cat: # membuat class Cat
    def meow(self): # fungsi meow didalam class Cat
        print('meowwww')

nobi = Cat() # membuat sebuah instance dari Cat
nobi.meow() # setelah membuat object dari Cat, kita bisa memanggil
             # method meow

#output
meowwww
```

#### d. Constructor `__init__` Method

Dalam konsep OOP kita juga mengenal yang namanya *constructor*. Dimana *constructor* umumnya digunakan untuk membuat *instance* objek. Tugas *constructor* adalah menginisialisasi (menetapkan nilai) ke anggota data dari *class* tersebut ketika objek dari *class* tersebut dibuat.

```
class Cat:  
    def __init__(self, name, color): # menginisialisasi instance dengan  
    constructor  
        self.name = name  
        self.color = color  
  
nobi = Cat('nobi', 'black') # membuat instance dari kelas Cat dengan  
nama nobi dan warna hitam  
nero = Cat('nero', 'white')  
  
print(nobi.name)  
print(nobi.color)  
print(nero.name)  
print(nero.color)  
  
#output  
nobi  
black  
nero  
white
```



#### e. Instance vs Class Variables

Jika nilai dari variabel bervariasi dari objek ke objek, maka variabel tersebut disebut *instance variables*. *Instance variables* tidak dibagikan oleh objek. Setiap objek **memiliki salinan sendiri** dari *instance attribute*. Artinya, untuk setiap objek suatu kelas, memiliki nilai *instance variables* yang berbeda.

- Perlakuan Instance Variables:

```
class Circle:  
    def __init__(self, name, radius, PI):  
        self.__name = name # instance variable  
        self.__radius = radius # instance variable  
        self.__PI = PI  
  
    # menghitung area sebuah lingkaran dengan pi * r kuadrat  
    def area(self):  
        return self.__PI * self.__radius ** 2  
  
c1 = Circle("C1", 4, 3.14)  
print("Area dari c1:", c1.area())  
c2 = Circle("C2", 6, 3.141)  
print("Area dari c2:", c2.area())  
c3 = Circle("C3", 6, 3.1415)  
print("Area dari c3:", c3.area())  
  
#output
```



`__name`, `__radius`, `__PI` dalam kode di atas dapat memiliki nilai yang berbeda untuk setiap *instance*. Variabel seperti itu disebut *instance variables*.

Alasan menggunakan awalan '`_`' adalah untuk menyembunyikan *instance* ini dari akses eksternal.

- Perlakuan Class Variables:

Dari contoh di atas mengenai *instance variables*, terkadang kita **memerlukan suatu variabel yang bisa di bagikan ke seluruh *instance* dari *class* tersebut**, contohnya adalah nilai dari **PI**.

Nilai dari **PI** adalah *constant* sehingga seharusnya masing-masing *instance* memiliki **nilai PI yang sama** dan tidak perlu di deklarasikan di masing-masing *instance* serta mengurangi kemungkinan *error*. Maka disini kita menggunakan yang disebut sebagai *class variable*.

```
class Circle:  
    PI = 3.1415  
    def __init__(self, name, radius):  
        self.__name = name # instance variable  
        self.__radius = radius # instance variable  
  
        # menghitung area sebuah lingkaran dengan pi * r kuadrat  
        # method dari class Circle dan instance akan mengambil value PI dari  
        class variable melalui Circle.PI  
    def area(self):  
        return Circle.PI * self.__radius ** 2  
  
c1 = Circle("C1", 4)  
print("Area dari c1:", c1.area())  
c2 = Circle("C2", 6,)  
print("Area dari c2:", c2.area())  
c3 = Circle("C3", 5)  
print("Area dari c3:", c3.area())  
  
#output  
Area dari c1: 50.264  
Area dari c2: 113.09400000000001  
Area dari c3: 78.53750000000001
```

Dari contoh di atas, seluruh *instance* menggunakan *value PI* dari *class variables PI* yaitu **3.1415**.

## f. Inheritance

- Perlakuan Inheritance:



```
1 class Person:
2     name = 'test'
3
4 class Employee (Person):
5     gaji = 100
6
7 person1 = Person()
8 employee1 = Employee()
9
10 print(person1.name)
11 # print(person1.gaji) # jika parent, tidak akan bisa mengakses child
12 print(employee1.gaji)
13 print(employee1.name) # jika child, bisa mengakses class variable milik parent
```



## g. Polymorphism

Pada *child class*, selain mewarisi semua sifat dari *parent*-nya, *child class* juga bisa memiliki kemampuan yang berbeda dengan *parent*-nya. Dengan kata lain, *child class* memiliki metode yang sama dengan *parent*-nya namun bisa jadi berbeda *output* dengan *sibling* atau *parent*-nya, hal ini dikenal dengan sebutan **polymorphism**.

```
class Bird:
    def intro(self):
        print("There are different types of birds")

    def flight(self):
        print("Most of the birds can fly but some cannot")

class parrot(Bird):
    def flight(self):
        print("Parrots can fly")

class penguin(Bird):
    def flight(self):
```

```

        print("Penguins do not fly")

obj_bird = Bird()
obj_parr = parrot()
obj_peng = penguin()

obj_bird.intro()
obj_bird.flight()

obj_parr.intro()
obj_parr.flight()

obj_peng.intro()
obj_peng.flight()

# output
There are different types of birds
Most of the birds can fly but some cannot
There are different types of bird
Parrots can fly
There are many types of birds
Penguins do not fly

```



## h. Encapsulation

*Encapsulation* menawarkan cara bagi kita untuk \*\*mengakses attribute \*\*yang diperlukan **tanpa memberikan program akses penuh ke salah satu attribute** tersebut.

- Single Underscores:

```

class Orang:
    def __init__(self, name, age=0):
        self.name = name
        self._age = age

    def tampilan(self):
        print(self.name)
        print(self._age)

orang_obj = Orang('Budi', 30)

#mengakses melalui metode

```

```

orang_obj.tampilkan()

#mengakses langsung variable
print(orang_obj.name)
print(orang_obj._age)

#output
Budi
30
Budi
30

```

Terlihat dari kode di atas bahwa kita tetap bisa mengakses langsung variable `_age` dari luar *class* menggunakan *instance* dari *class* tersebut.

- Double Underscores:

Jika kita ingin menjadikan anggota *class* yaitu *method* dan *attribute* menjadi *private*, maka kita harus mengawalinya dengan garis bawah ganda atau *double underscore* `__`.



```

class Orang:
    def __init__(self, name, age=0):
        self.name = name
        self.__age = age

```

```

    def tampilkan(self):
        print(self.name)
        print(self.__age)

```

```

orang_obj = Orang('Budi', 30)

```

```

#mengakses melalui metode
orang_obj.tampilkan()

```

```

#mengakses langsung variable
print(orang_obj.name)
print(orang_obj.__age)

```

```

#output
Budi
30
Budi
Traceback (most recent call last):
  File "test.py", line 17, in <module>

```

```
    print(orang_obj.__age)
AttributeError: 'Orang' object has no attribute '__age'
```

Jika kita menggunakan *double underscore* pada variable `__age`, maka terlihat kita hanya bisa mengakses variabel tersebut melalui metode *getter* yaitu `tampilkan()` dan akan muncul *error* apabila kita ingin mencoba mengakses variabel tersebut diluar dari *class*.

Untuk mengubah data, kita juga perlu membuat *setter* karena tidak bisa mengubah data secara langsung.

```
class Orang:
    def __init__(self, name, age=0):
        self.name = name
        self.__age = age

    def setAge(self, age):
        self.__age = age

    def tampilkan(self):
        print(self.name)
        print(self.__age)
        print("-----")

orang_obj = Orang('Budi', 30)

#mengakses melalui metode
orang_obj.tampilkan()

#mengubah data tanpa setter tidak memungkinkan
orang_obj.__age = 50
orang_obj.tampilkan()

#mengubah data harus memalui setter
orang_obj.setAge(27)
orang_obj.tampilkan()

#output
Budi
30
-----
Budi
30
-----
Budi
27
```



## i. Abstraction

```
from abc import ABC, abstractmethod

class Animal(ABC):

    @abstractmethod
    def speak(self):
        pass

class Dog(Animal):
    # Tidak mengimplementasikan speak()
    def speak(self):
        print('dawg')

    # Ini akan menghasilkan error
dog = Dog()
dog.speak()

# INTINYA abstract method itu merupakan acuan untuk subclass dibawahnya, misal di abstract method ada method speak. nah berarti subclass di bawahnya harus memilikinya.
```

## 6. Parallelism

### a. Apa itu Concurrency dan Parallelism

Concurrency sering dipahami sebagai "mengelola" banyak pekerjaan secara bersamaan. Pada kenyataannya, pekerjaan-pekerjaan itu tidak benar-benar dijalankan pada saat yang bersamaan. Mereka bergantian secara *Parallelism*, berarti **mengeksekusi beberapa pekerjaan secara bersamaan atau secara paralel**. *Parallelism* memungkinkan untuk memanfaatkan banyak *core* pada satu mesin.

Untuk menerapkan kedua konsep tersebut dalam Python, kita bisa menggunakan *library*:

- `multiprocess` untuk *parallelism* dan,
- `threading` untuk *concurrency*



## b. Threading

*Thread* adalah entitas dalam proses yang dapat dijadwalkan untuk dieksekusi. *Thread* juga adalah unit pemrosesan terkecil yang dapat dilakukan dalam Sistem Operasi (*Operating System*).

*Multithreading* didefinisikan sebagai kemampuan prosesor untuk mengeksekusi beberapa **thread** secara bersamaan.



```
 1 # program threading sederhana
 2 import threading
 3 import time
 4
 5 # Mendeklarasikan fungsi yang akan dijalankan dalam thread
 6 def print_numbers(n):
 7     for i in range(n):
 8         time.sleep(0.1) # Simulasi proses dengan delay 1 detik
 9         print(f"Nomor: {i+1}")
10
11 # Membuat thread dan menjalankan fungsi dengan parameter
12 t1 = threading.Thread(target=print_numbers, args=(100,)) # Argumen 5 akan dikirimkan ke print_numbers
13 t1.start() # Memulai thread
14
15 # Menunggu thread t1 selesai sebelum melanjutkan ke baris berikutnya
16 t1.join()
17
18 print("Thread selesai!")
```

### c. Multiprocessing

**Multiprocessing** mengacu pada kemampuan sistem untuk mendukung lebih dari satu prosesor pada saat yang bersamaan. Aplikasi dalam sistem *multiprocessing* dipecah menjadi rutinitas yang lebih kecil yang berjalan secara independen. Sistem operasi mengalokasikan *thread* ini ke prosesor untuk meningkatkan kinerja sistem.

*Multiprocessing* sangat cocok untuk program atau sistem yang **membutuhkan komputasi CPU yang tinggi, karena bisa memaksimalkan jumlah CPU yang ada didalam komputer tersebut**. Namun Ada beberapa hal yang perlu diperhatikan dalam process *multiprocessing*.



```
● ● ●
1 import multiprocessing
2 import time
3
4 # Mendeklarasikan fungsi yang akan dijalankan dalam proses terpisah
5 def print_numbers(n):
6     for i in range(n):
7         time.sleep(1) # Simulasi proses dengan delay 1 detik
8         print(f"Nomor: {i+1}")
9
10 # Membuat proses dan menjalankan fungsi dengan parameter
11 p1 = multiprocessing.Process(target=print_numbers, args=(5,)) # Argumen 5 akan dikirimkan ke print_numbers
12 p1.start() # Memulai proses
13
14 # Menunggu proses p1 selesai sebelum melanjutkan ke baris berikutnya
15 p1.join()
16
17 print("Proses selesai!")
```

Perbedaan mendasar antara *multiprocessing* dengan *thread* dalam Python adalah *multiprocessing* secara efektif mengesampingkan Global Interpreter Lock atau (GIL) dengan menggunakan subproses, bukan menggunakan *thread*. Oleh karena itu, modul *multiprocessing* memungkinkan program untuk sepenuhnya memanfaatkan *multi-core processor* pada mesin atau komputer kita.

**Akhir Kata,**

