

BAB 2. DASAR PEMROGRAMAN PYTHON

Daftar Isi

- 2.1. TIPE DATA DI PYTHON
- 2.2. KEYWORD PADA PYTHON
- 2.3. OPERATOR
- 2.4. CONDITIONAL STATEMENT, LOOP, FUNCTION, EXCEPTION HANDLING
- 2.5. OPERASI FILE SYSTEM PADA PYTHON
- 2.6. OOP PADA PYTHON
- 2.7. CLOSURE, METAPROGRAMMING, DECORATOR DAN METACLASS
- 2.8. KUMPULAN TRIK DASAR PYTHON

© Copyright by Antonius – www.jasaplus.com All Rights Reserved

Barangsiapa didapati memperjual belikan materi training ini tanpa seizin dari pencipta maka pencipta berhak menuntut ganti rugi yang jumlahnya ditentukan oleh pencipta materi ini.

BAB 2. DASAR PEMROGRAMAN PYTHON

2.1. TIPE DATA DI PYTHON

Pada saat mempelajari suatu bahasa pemrograman, sangat penting untuk memahami tipe data yang ada pada bahasa pemrograman tersebut. Berikut ini adalah jenis tipe data built-in yang ada pada python :

boolean, number (numerik), string, list, tuple, dictionary, set.

BOOLEAN

Tipe data boolean adalah tipe data yang hanya memiliki kemungkinan 2 nilai kebenaran, bisa bernilai benar (True) atau bisa bernilai salah (False).

NUMBER

Tipe data numerik di python, merupakan tipe data yang digunakan untuk menyimpan nilai berupa numerik (angka). Berikut ini beberapa tipe data numerik di python3:

1. int

merupakan tipe data bernilai angka bisa berisi nilai minus, pada python3, tipe data berisi angka dengan range berapapun akan secara otomatis ditangani oleh **kelas int** di python3. Untuk lebih jelasnya perhatikan image berikut ini :

[illegible]

3. float

merupakan tipe data yang bernilai angka, bisa berisi nilai koma. Secara default semua jenis angka di python3 akan dianggap integer, kecuali dikonversi dengan fungsi float ke tipe float atau jika nilai berisi koma maka akan dianggap tipe data float. Perhatikan gambar di bawah ini :

```
>>> type(100.0)
<class 'float'>
>>> type(100.01)
<class 'float'>
>>> type(100)
<class 'int'>
>>>
```

Untuk mengkonversi dari nilai integer ke float, kita bisa menggunakan fungsi `float()`.

4. complex

merupakan tipe data yang berisi nilai kompleks. Terdiri dari angka real dan angka imajiner. Pola nilai kompleks pada python3 : `<angka real> + <angka imajiner>j` .

Untuk lebih jelasnya perhatikan gambar di bawah ini :

```
>>> print("complex sample by jasaplus.com")
complex sample by jasaplus.com
>>> type(2+3j)
<class 'complex'>
>>> z = complex(4+4)
>>> type(z)
<class 'complex'>
>>> print(z)
(8+0j)
>>> z = complex(4,4)
>>> print(z)
(4+4j)
>>> type(z)
<class 'complex'>
>>> █
```

STRING

String adalah struktur data di python yang digunakan untuk menyimpan data di mana string terdiri dari kumpulan karakter (atau bisa juga hanya merupakan satu karakter).

contoh string :

```
string1 = "dataku"
string2 = "2 x 2"
string3 = "a"
```

Python memiliki fasilitas untuk string formatting. Misal pada saat printing data, kita bisa menggunakan string formatting dengan menggunakan placeholder berupa : {}

Contoh:

```
#!/usr/bin/env python3
"""
string formatting sample for
python course at www.jasaplus.com
"""
if __name__ == "__main__":
    nombre = 100
    strdata = "hax"
    floatnum = 5.5

    print("I print {} then I print {} then I print {}".format(nombre, strdata, floatnum))
```

Pada python 3.6 terdapat cara untuk melakukan format string lebih baru dengan nama “f-strings”

LIST

List adalah struktur data di python yang merupakan koleksi data yang bisa diubah ubah dan berurutan.

contoh list :

```
list1 = ["satu", "dua", "tiga"]
list2 = [1,2,3]
```

TUPLE

Tuple adalah struktur data di python yang merupakan koleksi data yang bersifat tetap dan berurutan.

contoh tuple:

```
tuple1 = ("satu", "dua", "tiga")
```

DICTIONARY

Dictionary adalah struktur data di python yang merupakan koleksi dari item item yang tidak berurutan. Masing masing item dalam suatu dictionary di python terdiri dari key dan value. Dictionary merupakan koleksi yang bisa diubah ubah.

contoh dictionary:

```
dict1 = {"nama": "adam", "pekerjaan": "programmer"}
```

SET

Set adalah struktur data di python yang digunakan untuk menyimpan data, di mana set tidak berurutan, unik (tidak ada duplikat). Kita bisa menambah atau mengurangi elemen di dalam set.

contoh set :

```
set1 = {"a", "bc", "ex"}  
set2 = {1,2,3}
```

Selain itu, set juga bisa dibuat dengan menggunakan fungsi built-in set, contoh :

```
dat = set(["hacker", "cracker", "carder"])
```

FROZEN SET

Frozen set adalah set yang tidak bisa diubah ubah. Untuk membuat frozenset, kita bisa menggunakan fungsi built-in (bawaan) . Contoh :

```
fs = frozenset(["fruit", "burger", "cake", "pizza"])
```

RUANG LINGKUP (SCOPE) VARIABEL PADA PYTHON

Pada kesempatan kali ini, kita akan mempelajari tentang ruang lingkup (scope) dari variabel di python.

Di python 3 kita mengenal beberapa scope variabel :

1. Global

Variabel yang didefinisikan di luar fungsi secara otomatis akan dianggap variabel global di mana variabel tersebut bisa dibaca dari dalam fungsi. Selain itu variabel yang didefinisikan dari dalam suatu fungsi dengan keyword global juga akan dianggap variabel global. Jika pada saat definisi, variabel tersebut tidak pernah didefinisi dengan keyword global, maka secara otomatis akan dianggap global dan bisa dibaca dari dalam fungsi, akan tetapi jika variabel tersebut telah dibaca sekali dari dalam fungsi, pada baris selanjutnya di dalam fungsi variabel tersebut tidak bisa diubah

lagi, variabel tersebut bisa kita ubah jika kita definisikan di baris paling atas fungsi, atau bisa juga didefinisikan dengan keyword global.

2. Local

Variabel yang didefinisikan di dalam suatu fungsi tanpa keyword global akan dianggap variabel local di mana variabel tersebut hanya bisa dibaca dari dalam fungsi itu sendiri.

3. Non Local

nonlocal variable adalah jenis variabel yang digunakan hanya khusus pada fungsi di dalam fungsi di mana skope variabel tersebut hanya bisa dibaca pada fungsi di dalam suatu fungsi.

Aturan Global dan Local

Beberapa aturan tentang global dan local variable. Untuk lebih jelasnya, pertama tama buat script python dengan nama global_local.py dengan isi di bawah ini :

```
#!/usr/bin/env python3
"""
global_local.py
sample source code for python training
at jasaplus.com
"""
globalone = "hello"
globalnum = 77

def myfunc():
    globalone = "wtf"
    print("globalone : " + globalone)
    print("globalnum : ", globalnum)
    hack = "200"
    print(hack)

def defglo():
    global hack
    hack = "123"

if __name__ == "__main__":
    defglo()
    myfunc()
```

Pada contoh di atas, variabel globalone dan globalnum dideklarasikan di luar fungsi, sehingga secara default memiliki akses skope global.

```
globalone = "hello"
globalnum = 77
```

Pada fungsi defglo, kita memiliki satu global variabel : "hack" :

```
def defglo():
    global hack
    hack = "123"
```

Jika aplikasi di atas dijalankan maka akan tampil :

```
globalone : wtf
globalnum : 77
200
```

```
if __name__ == "__main__":
    def glo()
    myfunc()
```

Pada saat script dijalankan akan mengeksekusi fungsi def glo di mana variabel global hack didefinisikan, selanjutnya fungsi myfunc akan dijalankan, di mana fungsi myfunc berisi:

```
def myfunc():
    globalone = "wtf"
    print("globalone : " + globalone)
    print("globalnum : ", globalnum)
    hack = "200"
    print(hack)
```

Terlihat saat menjalankan fungsi myfunc variabel globalone dan globalnum bisa dibaca karena dianggap variabel global.

Kembali lagi ke kutipan keterangan tentang variabel global : *"Variabel yang didefinisikan di luar fungsi secara otomatis akan dianggap variabel global di mana variabel tersebut bisa dibaca dari dalam fungsi. Selain itu variabel yang didefinisikan dari dalam suatu fungsi dengan keyword global juga akan dianggap variabel global. Jika pada saat definisi, variabel tersebut tidak pernah didefinisi dengan keyword global, maka secara otomatis akan dianggap global dan bisa dibaca dari dalam fungsi, akan tetapi jika variabel tersebut telah dibaca sekali dari dalam fungsi, pada baris selanjutnya di dalam fungsi variabel tersebut tidak bisa diubah lagi, variabel tersebut bisa kita ubah jika kita definisikan di baris paling atas fungsi, atau bisa juga didefinisikan dengan keyword global."*

Untuk lebih jelasnya, kita buat file baru dengan nama global_local2.py :

```
#!/usr/bin/env python3
"""
global_local2.py
sample buggy source code for python training
at jasaplus.com
"""
globalone = "hello"
globalnum = 77

def myfunc():
    print("globalone : " + globalone)
    print(type(globalone))
    print("globalnum : ", globalnum)
    hack = "200"
    print(hack)
    globalone = "modified"
    print(globalone)

if __name__ == "__main__":
    myfunc()
```

Perhatikan ! Pada source di atas ada kesalahan di mana variabel globalone diinisialisasi setelah pembacaan sebelumnya pada baris di atasnya:

```
print("globalone : " + globalone)
```

Aturannya adalah variabel global yang berasal dari luar fungsi, setelah dibaca 1x pada baris di atas, tidak bisa diubah pada baris di bawahnya kecuali didefinisi ulang pada awal fungsi.

Jika script di atas dijalankan maka akan muncul pesan error :

```
ringlayer@ringlayer-Inspiron-3442:~/Desktop/TRAINING/PYTHON/source/variabel$ ./global_local2.py
Traceback (most recent call last):
  File "./global_local2.py", line 20, in <module>
    myfunc()
  File "./global_local2.py", line 11, in myfunc
    print("globalone : " + globalone)
UnboundLocalError: local variable 'globalone' referenced before assignment
ringlayer@ringlayer-Inspiron-3442:~/Desktop/TRAINING/PYTHON/source/variabel$
```

Kesalahan terjadi saat kita mendefinisikan variabel globalone setelah pembacaan terjadi

```
globalone = "modified"
```

Untuk contoh yang benar, buat aplikasi baru dengan nama global_local3.py dengan isi :

```
#!/usr/bin/env python3
"""
global_local3.py
sample correct source code for python training
at jasaplus.com
"""
globalone = "hello"
globalnum = 77

def myfunc():
    global globalone
    print("globalone : " + globalone)
    print(type(globalone))
    print("globalnum : ", globalnum)
    hack = "200"
    print(hack)
    globalone = "modified"
    print(globalone)

if __name__ == "__main__":
    myfunc()
```

Pada contoh di atas, di awal fungsi kita definisikan variabel globalone sebagai variabel global, dengan cara ini kita bisa mengakses variabel globalone sebagai variabel dengan scope global tanpa mempedulikan aturan penyebab kesalahan sebelumnya.

Hasil dari menjalankan aplikasi di atas :

```
ringlayer@ringlayer-Inspiron-3442:~/Desktop/TRAINING/PYTHON/source/variabel$ ./global_local3.py
globalone : hello
<class 'str'>
globalnum : 77
200
modified
ringlayer@ringlayer-Inspiron-3442:~/Desktop/TRAINING/PYTHON/source/variabel$
```

Non Local Variable ?

Scope dari variabel nonlocal hanya berlaku untuk fungsi di dalam suatu fungsi. Perhatikan contoh pada nested_def.py berikut ini :

```
#!/usr/bin/env python3

"""
nested_def.py
sample nested def with nonlocal
python sample app for course material on jasaplus.com
"""

def uji():
    print("fungsi dalam fungsi")
    variabel = "hack"
    paramet = "parameter"
    def dalam1(param):
        print("dalaman 1")
        print(param)

    def dalam2():
        nonlocal variabel
        print(variabel)
        variabel = "changed"

    def dalam3():
        print("3rd inner def")

    dalam3()
    print(variabel)
    dalam2()
    print(variabel)

    dalam1(paramet)

if __name__ == "__main__":
    uji()
```

Hasil dari script di atas :



```
ringlayer@ringlayer-Inspiron-3442:~/Desktop/github/PYTHON/nested_def$ python3 nested_def.py
fungsi dalam fungsi
3rd inner def
hack
hack
changed
dalaman 1
parameter
ringlayer@ringlayer-Inspiron-3442:~/Desktop/github/PYTHON/nested_def$
```

variabel “variabel” pada fungsi inner dalam2() dianggap sebagai variabel non lokal milik fungsi parent uji()).

2.2. KEYWORD PADA PYTHON

Keyword adalah kata yang dicadangkan di python, yang tidak bisa digunakan sebagai nama variabel, fungsi, atau identifier lainnya.

Untuk melihat daftar keyword pada python, pada python console:

```
>>> import keyword
>>> print(keyword.kwlist)
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except',
'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return',
'try', 'while', 'with', 'yield']
```

Pada console python di atas terlihat keyword keyword yang digunakan pada python.

False

keyword ini digunakan untuk nilai kebenaran salah (tipe data boolean)

None

keyword ini digunakan untuk mengisi nilai kosong ke suatu variabel.

True

keyword ini digunakan untuk nilai kebenaran benar (tipe data boolean)

and

keyword ini digunakan untuk membandingkan 2 operand / statement. Keyword ini akan menghasilkan nilai True jika keduanya bernilai True.

as

keyword ini digunakan untuk membuat suatu alias.

assert

keyword ini berfungsi untuk debugging aplikasi. Keyword ini akan menguji suatu kondisi pada kode apakah menghasilkan nilai True, jika tidak maka assert gagal dan akan menghasilkan error AssertionError.

break

keyword ini digunakan untuk keluar dari suatu blok pengulangan.

class

keyword ini digunakan untuk menciptakan suatu class di python.

continue

keyword ini digunakan untuk mengakhiri satu iterasi pada pengulangan, keyword ini akan menyebabkan aplikasi melanjutkan ke iterasi selanjutnya.

def

keyword ini digunakan untuk membuat / mendefinisikan suatu fungsi atau method.

del

keyword ini digunakan untuk menghapus suatu objek / variabel.

elif

keyword ini digunakan untuk memulai suatu pernyataan pengujian kondisi, keyword ini pada dasarnya merupakan singkatan / bentuk pendek dari else if

else

keyword ini digunakan pada saat pengujian kondisi dengan if, di mana jika hasil yang diuji dari if bernilai false maka akan mengeksekusi blok kode pada else.

except

keyword ini digunakan bersamaan dengan penanganan error dengan try. Di mana pola penanganan error ini memiliki pola : try ... except, di mana blok kode pada try adalah blok kode yang akan diuji, jika terdapat error pada blok kode try, maka blok kode pada except akan dieksekusi.

finally

keyword ini digunakan di bawah blok kode try...except, di mana blok kode pada finally adalah blok kode yang pasti akan dieksekusi saat blok pada try menghasilkan error atau tidak.

for

keyword ini digunakan untuk membuat pengulangan.

from

keyword ini digunakan untuk mengimport hanya suatu bagian dari modul.

global

keyword ini digunakan untuk mendefinisikan variabel yang bersifat global.

if

keyword ini digunakan untuk memulai suatu blok pengujian kondisi.

import

keyword ini digunakan untuk mengimpor modul di python ke aplikasi yang kita buat.

in

keyword ini digunakan untuk menguji apakah suatu nilai ada pada suatu sequence (list, range, string, dll). Keyword ini juga bisa digunakan bersamaan dengan pengulangan dengan for.

is

keyword ini digunakan untuk menguji apakah dua objek memiliki referensi yang sama. Pada dasarnya keyword "is" bisa digunakan untuk menguji apakah satu objek memiliki lebih dari satu nama. "is" digunakan untuk menguji apakah kedua objek yang dijadikan operand adalah identik.

lambda

keyword ini digunakan untuk membuat fungsi tanpa nama (anonymous function)

nonlocal

keyword ini digunakan untuk penggunaan variabel pada fungsi di dalam fungsi. Penggunaan keyword nonlocal pada definisi suatu variabel di dalam inner def (fungsi inner / fungsi di dalam fungsi) akan menyebabkan variabel tersebut dianggap variabel milik class parent.

not

keyword ini digunakan untuk menguji nilai suatu variabel, akan menghasilkan nilai True jika variabel bukan True.

or

keyword ini merupakan keyword yang digunakan sebagai operator logika. Keyword ini akan menguji logika or pada operand kiri dan operand kanan.

pass

keyword ini digunakan untuk baris kode, di mana baris tersebut tidak melakukan apapun. Keyword ini bisa digunakan untuk membuat fungsi kosong atau bisa digunakan saat blok try...except di mana baris pada blok di dalam except tidak melakukan apapun.

raise

keyword ini digunakan untuk menghasilkan exception (error handling).

return

keyword ini digunakan untuk keluar dari suatu fungsi dan mengembalikan suatu nilai dari fungsi.

try

keyword ini digunakan untuk menguji suatu blok kode terhadap terjadinya kemungkinan error.

while

keyword ini digunakan untuk memulai suatu blok pengulangan dengan while.

with

keyword ini digunakan untuk error handling (penanganan error) serupa dengan blok kode try. Keyword ini merupakan keyword yang biasa digunakan untuk penanganan file descriptor di mana setelah blok kode selesai dieksekusi file descriptor tersebut secara otomatis akan ditutup.

yield

keyword ini digunakan seperti return statement di dalam fungsi, tetapi keyword ini bukan menghasilkan nilai kembali seperti pada return, keyword ini akan mengembalikan generator.

2.3. OPERATOR

Operator adalah simbol khusus di python yang digunakan untuk melakukan operasi pada variabel dan nilai.

Jenis jenis operator pada python:

1. Operator Perhitungan (Arithmetic)

Digunakan untuk operasi perhitungan. Contoh operator perhitungan:

Operator +

Digunakan untuk operasi penjumlahan

Operator -

Digunakan untuk operasi pengurangan

Operator *

Digunakan untuk operasi perkalian

Operator /

Digunakan untuk operasi pembagian

Operator %

Digunakan untuk operasi modulus

Operator **

Digunakan untuk operasi eksponensial

Operator //

Digunakan untuk operasi pembagian dengan pembulatan ke bawah

2. Operator Penunjukan (Assignment)

Digunakan untuk pengisian nilai suatu variabel. Berikut ini operator operator penunjukan pada python:

=

Digunakan untuk mengisi nilai pada operand sebelah kanan ke operand sebelah kiri.

+=

Digunakan untuk menambah jumlah pada operand kiri sejumlah operand sebelah kanan.

-=

Digunakan untuk mengurangi jumlah pada operand kiri sejumlah operand sebelah kanan.

***=**

Digunakan untuk mengalikan jumlah pada operand kiri sejumlah operand sebelah kanan.

/=

Digunakan untuk membagi jumlah pada operand kiri sejumlah operand sebelah kanan.

%=

Digunakan untuk operasi modulus untuk operand kiri di mana pembaginya adalah operand kanan. Sisa pembagiannya (modulus) disimpan pada operand kiri.

//=

Digunakan untuk membagi dengan pembulatan ke bawah. Di mana operand kiri akan dibagi operand kanan kemudian hasilnya dibulatkan ke bawah dan disimpan pada operand kiri.

****=**

Digunakan untuk operasi pangkat di mana operand kiri akan dipangkatkan operand kanan dan hasilnya disimpan pada operand kiri.

Operator assignment selanjutnya berhubungan dengan operasi bilangan biner.

&=

Pada dasarnya & adalah operator bitwise and, &= merupakan perintah untuk melakukan ini: operand sebelah kiri akan dilakukan bitwise and dengan operand kanan, selanjutnya hasilnya akan diassign / disimpan pada operand kiri

|=

Pada dasarnya | adalah operator bitwise or, |= merupakan perintah untuk melakukan ini: operand sebelah kiri akan dilakukan bitwise or dengan operand kanan, selanjutnya hasilnya akan diassign / disimpan pada operand kiri

^=

Pada dasarnya ^ adalah operator bitwise exclusive or, ^= merupakan perintah untuk melakukan ini: operand sebelah kiri akan dilakukan bitwise exclusive or dengan operand kanan, selanjutnya hasilnya akan diassign / disimpan pada operand kiri

>>=

Pada dasarnya >> adalah operator bitwise, >>= akan menyebabkan nilai pada operand kiri digeser ke kanan sebesar nilai pada operand kanan.

<<=

Pada dasarnya << adalah operator bitwise, <<= akan menyebabkan nilai pada operand kiri digeser ke kiri sebesar nilai pada operand kanan.

3. Operator Perbandingan (Comparison)

Digunakan untuk perbandingan 2 operand. Berikut ini operator operator perbandingan pada python :

==

Merupakan operator perbandingan di mana operand kiri akan dibandingkan nilainya dengan operand kanan, jika nilainya sama maka hasilnya adalah true

!=

Merupakan operator perbandingan di mana operand kiri akan dibandingkan nilainya dengan operand kanan, jika nilainya tidak sama maka hasilnya adalah true

>

Merupakan operator perbandingan di mana operand kiri akan dibandingkan nilainya dengan operand kanan, jika nilainya operand kiri lebih besar dari operand kanan maka hasilnya adalah true

<

Merupakan operator perbandingan di mana operand kiri akan dibandingkan nilainya dengan operand kanan, jika nilainya operand kiri lebih kecil dari operand kanan maka hasilnya adalah true

>=

Merupakan operator perbandingan di mana operand kiri akan dibandingkan nilainya dengan operand kanan, jika nilainya operand kiri lebih besar atau sama dengan operand kanan maka hasilnya adalah true

<=

Merupakan operator perbandingan di mana operand kiri akan dibandingkan nilainya dengan operand kanan, jika nilainya operand kiri lebih kecil atau sama dengan operand kanan maka hasilnya adalah true

4. Operator Logika (Logical)

Operator ini digunakan untuk operasi perbandingan logika untuk operand. Berikut ini contoh operator logika pada python :

and

Merupakan operator untuk perbandingan nilai logika. Akan menghasilkan nilai true jika operand sebelah kiri bernilai true dan operator sebelah kanan bernilai true.

or

Merupakan operator untuk perbandingan nilai logika. Akan menghasilkan nilai true jika operand sebelah kiri bernilai true atau operator sebelah kanan bernilai true.

not

Merupakan operator logika yang akan mengembalikan nilai true jika operand bernilai false.

5. Operator Identitas (Identity)

Berikut ini operator identitas pada python:

is

Akan menghasilkan nilai true jika kedua operand identik

is not

Akan menghasilkan nilai true jika kedua operand tidak identik

6. Operator Keanggotaan (Membership)

Berikut ini operator keanggotaan pada python:

in

Operator ini digunakan untuk menguji apakah operand ditemukan pada sequence seperti list, tuple, set, string. Jika ditemukan maka akan menghasilkan nilai true.

not in

Operator ini digunakan untuk menguji apakah operand tidak ditemukan pada sequence seperti list, tuple, set, string. Jika tidak ditemukan maka akan menghasilkan nilai true.

7. Operator Bitwise

Operator bitwise digunakan untuk operasi pada level bit. Berikut ini operator operator bitwise pada python :

&

Merupakan operator bitwise and di mana nilai kedua operand akan di AND

|

Merupakan operator bitwise or di mana nilai kedua operand akan di OR

^

Merupakan operator bitwise xor di mana nilai kedua operand akan di XOR

~

Merupakan operator bitwise not.

<<

Operator bitwise left, << akan menyebabkan nilai pada operand kiri digeser ke kanan sebesar nilai pada operand kanan.

>>

Operator bitwise right, >> akan menyebabkan nilai pada operand kiri digeser ke kanan sebesar nilai pada operand kanan.

2.4. CONDITIONAL STATEMENT, LOOP, FUNCTION, EXCEPTION HANDLING

CONDITIONAL STATEMENT

Pada python, pengujian kondisi bisa dilakukan dengan : IF, ELSE, ELIF atau Nested IF.

IF

Contoh penggunaan if :

```
#!/usr/bin/env python3
"""
if_sample.py
sample if conditional for course training
at www.jasaplus.com
"""
b = 100
def ifSample(a):
    global b
    if a > b :
        print("a larger than b")
if __name__ == "__main__":
    ifSample(200)
```

ELSE

Contoh penggunaan else :

```
#!/usr/bin/env python3
"""
if_else_sample.py
sample if else conditional for course training
at www.jasaplus.com
"""
b = 100
def ifSample(a):
    global b
    if a > b :
        print("a larger than b")
    else:
        print("b larger than a")
if __name__ == "__main__":
    ifSample(200)
    ifSample(90)
```

ELIF

Contoh penggunaan elif :

```
#!/usr/bin/env python3
"""
elif_sample.py
sample elif conditional for course training
at www.jasaplus.com
"""
```



```

password1 = "ringlayer"
password2 = "hack"

def Validate_Password(given_password):
    global password1
    global password2
    retme = 0
    if (given_password == password1):
        retme = 1
    elif (given_password == password2):
        retme = 2
    else:
        retme = -1

    return retme

if __name__ == "__main__":
    inputpass = input("password : ")
    result_privilege = Validate_Password(inputpass)
    if result_privilege == 1:
        print("admin privilege")
    elif result_privilege == 2:
        print("operator privilege")
    else:
        print("[-] error ! invalid password !")

```

NESTED IF

Nested if adalah conditional if yang berada di dalam blok conditional if.

Contoh nested if :

```

#!/usr/bin/env python3
'''
nested_if_sample.py
sample nested if conditional for course training
at www.jasaplus.com
'''

if __name__ == "__main__":
    orignum = 100
    word1 = "ab"
    word2 = "cd"
    word3 = "ef"
    given_number = int(input("type a number : ").strip())
    given_word = input("type a word : ").strip()
    if given_number > orignum:
        #nested if
        if given_word == word1:
            print("correct")
        elif given_word == word2:
            print("nearly correct")
        elif given_word == word3:
            print("nearly incorrect")
        else:
            print("fatal error ! program can't continue")
    else:
        print("fatal error ! application crash !")

```

LOOP

Pada python untuk melakukan pengulangan (loop), kita bisa menggunakan keyword : for, while.

FOR

Contoh penggunaan for :

```
#!/usr/bin/env python3
"""
for_sample.py
sample nested if conditional for course training
at www.jasaplus.com
"""
def forRepeat(num):
    print(num)

for a in range(5,15):
    forRepeat(a)
```

WHILE

Contoh penggunaan while :

```
#!/usr/bin/env python3
"""
while_sample.py
sample while for python training
at www.jasaplus.com
"""

def repeatme():
    print("repeated")

if __name__ == "__main__":
    start = 1
    max = 10
    while (start < max):
        print("current start : ", start)
        repeatme()
        start += 1
```

FUNCTION & METHOD

Fungsi pada python merupakan suatu blok kode yang digunakan untuk melakukan suatu tugas. Fungsi ini bisa dipanggil dari blok kode lain. Untuk membuat suatu fungsi pada python, kita menggunakan keyword “def”.

Method pada python adalah fungsi yang ada pada suatu class python. Berikut ini adalah contoh fungsi pada python, buat file baru dengan nama function_sample.py

```
#!/usr/bin/env/python3
"""
function_sample.py

sample function for python training
at www.jasaplus.com
```

```

'''
def function1():
    x = 2
    y = 3
    print("Sample function 1")
    z = x + y

    return z

def function2(param):
    print(param)

if __name__ == "__main__":
    function1()
    function2("we give function param here")

```

EXCEPTION HANDLING

python memiliki fitur untuk exception handling (penanganan exception). Exception adalah error / kesalahan program yang terjadi pada saat eksekusi suatu program. Python memiliki fitur exception handling untuk mendebug aplikasi di mana penyebab error dari jalanya aplikasi bisa dimunculkan saat exception handling. Selain itu exception handling juga bisa digunakan untuk mencegah aplikasi crash / berhenti total saat error.

Berikut ini beberapa macam error yang umum terjadi :

1. IOError

Terjadi saat file tidak bisa dibuka atau error yang berhubungan dengan input dan output file descriptor pada file system di sistem operasi.

2. IndexError

Error ini muncul ketika index tidak ditemukan pada sequence (contoh : list, tuple).

3. ImportError

Terjadi saat perintah import gagal mengimport modul python

4. ValueError

Terjadi ketika fungsi built in di python menerima argumen berupa variabel dengan nilai yang salah, misal seharusnya argumen berupa variabel tipe string tetapi variabel diisi dengan angka.

5. KeyboardInterrupt

Terjadi ketika user menekan tombol interrupt saat jalanya aplikasi, misal : ctrl+c

6. UnboundLocalError

Terjadi ketika mencoba mengakses variabel lokal pada suatu fungsi di mana variabel tersebut belum pernah diisi sebelumnya.

Untuk memunculkan exception saat terjadi error, kita gunakan keyword “raise”.

Blok Try ... Except

Metode yang sering digunakan untuk penanganan exception adalah dengan menggunakan blok kode try ... except.

Berikut ini contoh exception handling dengan try ... except :

```
#!/usr/bin/env python3
'''
rmdir_sample.py
sample remove a directory for course training
at www.jasaplus.com
'''
import os
if __name__ == "__main__":
    try:
        if os.path.isdir("directory_to_create"):
            os.rmdir("directory_to_create")
    except:
        raise
```

Pada contoh di atas terdapat try ... except block berikut ini :

```
try:
    if os.path.isdir("directory_to_create"):
        os.rmdir("directory_to_create")
except:
    raise
```

di mana pertama tama aplikasi akan mencoba mengecek keberadaan direktori, jika ditemukan maka akan dihapus dengan rmdir, jika terjadi exception maka akan dilakukan raise pesan error default yang muncul.

assert

Assert bisa digunakan untuk debuggin code python

finally

Pada bagian penjelasan keyword python, sudah diterangkan tentang finally. Finally ditempatkan setelah blok except, di mana blok finally adalah rutin yang sudah pasti akan dieksekusi.

2.5. OPERASI FILE SYSTEM PADA PYTHON

Pada bagian ini kita akan mempelajari operasi file system pada python.

2.5.1. Operasi Direktori

Pada bagian ini kita akan membuat beberapa contoh aplikasi di python yang berhubungan dengan operasi direktori.

Mengecek Keberadaan Direktori

Untuk mengecek keberadaan direktori, kita bisa menggunakan method `isdir()` di mana parameter fungsi ini adalah nama direktori yang ingin dicek keberadaanya. Berikut ini contoh penggunaan `isdir`, buat file baru dengan nama `isdir_sample.py` :

```
#!/usr/bin/env python3
"""
isdir_sample.py
sample checking whether a directory exists or not for course training
at www.jasaplus.com
"""
import os

if __name__ == "__main__":
    try:
        if os.path.isdir("directoryname"):
            print("exists")
        else:
            print("not found")
    except:
        raise
```

Membuat Direktori Baru

Untuk membuat direktori baru, kita akan menggunakan library `os`, di mana kita akan menggunakan method `mkdir` dari library `os`.

Sintaks :

```
os.mkdir("dirname")
```

Berikut ini adalah contoh aplikasi yang membuat direktori baru, buat file baru dengan nama `mkdir_sample.py` :

```
#!/usr/bin/env python3
"""
mkdir_sample.py
sample create directory for course training
at www.jasaplus.com
"""
import os

if __name__ == "__main__":
    try:
```

```

    if os.path.isdir("directoryname") == False:
        os.mkdir("directoryname")
except:
    raise

```

Menghapus Direktori

Untuk menghapus direktori kosong kita bisa menggunakan fungsi `rmdir()` dari library `os`. Buat file baru dengan nama `rmdir_sample.py`

```

#!/usr/bin/env python3
"""
rmdir_sample.py
sample remove a directory for course training
at www.jasaplus.com
"""
import os

if __name__ == "__main__":
    try:
        if os.path.isdir("directoryname"):
            os.rmdir("directoryname")
    except:
        raise

```

Untuk menghapus direktori tidak kosong, kita bisa menggunakan modul `shutil`. Kita bisa menggunakan method `rmtree` :

```

#!/usr/bin/env python3
"""
rmtree_sample.py
sample remove a non empty directory for course training
at www.jasaplus.com
"""
import os, shutil

if __name__ == "__main__":
    try:
        if os.path.isdir("non_empty_dir"):
            shutil.rmtree("non_empty_dir")
    except:
        raise

```

Melihat Daftar File dan Direktori di dalam Suatu Direktori

Untuk melihat daftar file dan direktori di dalam suatu direktori, kita bisa menggunakan fungsi `os.walk()` atau fungsi `os.listdir()`

Berikut ini adalah contoh aplikasi dengan method `os.walk()` dan `os.listdir()`, buat aplikasi baru dengan nama file `list_sample.py` dengan isi :

```

#!/usr/bin/env python3
"""
list_sample.py
sample source code for python training
at jasaplus.com
"""
from os import listdir, walk

```

```
def listdir_test(dirname):
    return [f for f in listdir(dirname)]

def oswalk_sample(dirname):
    return [f[0] for f in walk(dirname)]

if __name__ == "__main__":
    dirname = "/tmp"
    list1 = listdir_test(dirname)
    list2 = oswalk_sample(dirname)
    for f in list2:
        print(f)
    print("=" * 166)
    for f in list1:
        print(f)
```

Jika dijalankan, hasilnya adalah :

```
/tmp/Seanergyaar_cache
/tmp/.X11-unix
/tmp/.Test-unix
/tmp/.font-unix
/tmp/lu5527wuz7rn.tmp
/tmp/hsperfdata_ringlayer
/tmp/.ICE-unix
/tmp/ssh-JnqySMzhskUM
/tmp/mozilla_ringlayer0
/tmp/Atom Crashes
/tmp/Temp-dfce3db8-823d-4dd8-9437-502333d46fc6
/tmp/.org.chromium.Chromium.cDhI4r
/tmp/Temp-2f4794b7-9745-4c03-b6b6-65862f09be6f
/tmp/.XIM-unix
=====
+~JF8931072697468315583.tmp
+~JF3394030724055869605.tmp
+~JF3955057396147875677.tmp
systemd-private-97307afca7a3400aa2914627517ce27c-dovecot.service-stWQqz
ssh-F3U380IjJlZA
Seanergyaar_cache
qipc_systemSem_soliddiskinfomemac5ffa537fd8798875c98e190df289da7e047c05
+~JF8748489475357271007.tmp
.X11-unix
atom-Xzk-Tqlu_bmd.sock
+~JF4547554337734520752.tmp
systemd-private-97307afca7a3400aa2914627517ce27c-colord.service-WsC5Sw
.Test-unix
.font-unix
.X0-lock
.org.chromium.Chromium.awNwyr
.org.chromium.Chromium.en2V9G
+~JF2057289699168715994.tmp
lu5527wuz7rn.tmp
.org.chromium.Chromium.HjfhSC
OSL_PIPE_1000_SingleOfficeIPC_55dcee19e9e6b83355d3bcec896111
```

Penjelasan

Fungsi listdir_test akan mengembalikan list berupa daftar file pada dirname

```
def listdir_test(dirname):
    return [f for f in listdir(dirname)]
```

Fungsi oswalk_sample akan mengembalikan list berupa daftar file beserta path lengkap pada dirname

```
def oswalk_sample(dirname):
    return [f[0] for f in walk(dirname)]
```

Pada contoh di atas kita

Selanjutnya kita cek pada main :

```
if __name__ == "__main__":
    dirname = "/tmp"
    list1 = listdir_test(dirname)
    list2 = oswalk_sample(dirname)
    for f in list2:
        print(f)
    print("=" * 166)
    for f in list1:
        print(f)
```

Pada kedua baris ini :

```
list1 = listdir_test(dirname)
list2 = oswalk_sample(dirname)
```

list1 dan list2 masing masing akan berisi data berupa list dari fungsi listdir_test dan oswalk_sample.

Selanjutnya :

```
for f in list2:
    print(f)
```

Kode di atas akan melakukan looping di mana variabel bertipe string “f” akan diinisialisasi sebagai masing masing member dari list “list2”, di dalam blok loop, aplikasi akan melakukan printing data string “f”

```
for f in list1:
    print(f)
```

Mengkopi Direktori, Rename

Untuk mengkopi direktori, kita bisa menggunakan method copytree dari modul shutil, sedangkan untuk mengganti nama direktori, kita bisa menggunakan method rename dari modul os.

Selanjutnya, kita akan mencoba membuat aplikasi untuk mengkopi direktori dan mengganti nama direktori. Buat aplikasi baru dengan nama kopren_sample.py :

```
#!/usr/bin/env python3
"""
kopren_sample.py
sample source code for python training
at jasaplus.com
"""
import os, shutil

def _copytree(dirname, newdir):
    try:
        print("[+] copytree")
        shutil.copytree(dirname, newdir)
    except:
        raise

def _mkdir(dirname):
```



```

try:
    print("[+] mkdir")
    os.mkdir(dirname)
except:
    raise

def _rename(dirname, newname):
    try:
        print("[+] rename")
        os.rename(dirname, newname)
    except:
        raise

def _rmdir(dirname):
    try:
        print("[+] rmdir")
        os.rmdir(dirname)
    except:
        try:
            shutil.rmtree(dirname)
        except:
            raise
        pass

def func_kopidir(dirname, newdir):
    try:
        if os.path.isdir(newdir):
            print("[+] already copied before")
        elif os.path.isdir(dirname):
            _copytree(dirname, newdir)
        else:
            _mkdir(dirname)
            _copytree(dirname, newdir)
    except:
        raise

def func_rendir(dirname, newname):
    try:
        if (os.path.isdir(dirname)):
            if (os.path.isdir(newname) == False):
                _rename(dirname, newname)
            else:
                _rmdir(newname)
                _rename(dirname, newname)
        else:
            print("[-] directory {} not exists ... creating directory {} ".format(dirname, dirname))
            _mkdir(dirname)
            if (os.path.isdir(newname) == False):
                _rename(dirname, newname)
            else:
                _rmdir(newname)
                _rename(dirname, newname)
    except:
        raise

if __name__ == "__main__" :
    dirname = "test"
    newdir = "copied"
    newname = "changed"
    func_kopidir(dirname, newdir)
    func_rendir(dirname, newname)

```

Penjelasan

Fungsi `_copytree` merupakan fungsi wrapper untuk method `copytree`

```
def _copytree(dirname, newdir):
    try:
        print("[+] copytree")
        shutil.copytree(dirname, newdir)
    except:
        raise
```

Fungsi `_mkdir` merupakan fungsi wrapper untuk method `mkdir`

```
def _mkdir(dirname):
    try:
        print("[+] mkdir")
        os.mkdir(dirname)
    except:
        raise
```

Fungsi `_rename` merupakan fungsi wrapper untuk method `rename`

```
def _rename(dirname, newname):
    try:
        print("[+] rename")
        os.rename(dirname, newname)
    except:
        raise
```

Fungsi `_rmdir` merupakan fungsi wrapper untuk method `rmdir` :

```
def _rmdir(dirname):
    try:
        print("[+] rmdir")
        os.rmdir(dirname)
    except:
        try:
            shutil.rmtree(dirname)
        except:
            raise
    pass
```

Fungsi `func_rendir` merupakan fungsi yang digunakan untuk pekerjaan utama mengganti nama suatu direktori.

```
def func_rendir(dirname, newname):
    try:
        if (os.path.isdir(dirname)):
            if (os.path.isdir(newname) == False):
                _rename(dirname, newname)
            else:
                _rmdir(newname)
                _rename(dirname, newname)
        else:
            print("[-] directory {} not exists ... creating directory {}".format(dirname, dirname))
            _mkdir(dirname)
            if (os.path.isdir(newname) == False):
```

```

        _rename(dirname, newname)
    else:
        _rmdir(newname)
        _rename(dirname, newname)
except:
    raise

```

Untuk mempermudah penjelasan, perhatikan potongan kode ini :

```

if (os.path.isdir(dirname)):
    if (os.path.isdir(newname) == False):
        _rename(dirname, newname)
    else:
        _rmdir(newname)
        _rename(dirname, newname)

```

Cara kerja potongan kode pada fungsi `func_rendir` di atas : pertama tama aplikasi akan melakukan pengecekan apakah ada direktori “dirname”, jika tidak ditemukan maka aplikasi akan melakukan pengecekan apakah ada direktori dengan nama “newname”. Jika direktori “newname” tidak ditemukan maka direktori “dirname” akan langsung direname menjadi “newname”, sebaliknya jika direktori “newname” sudah ada sebelumnya maka direktori “newname” akan dihapus, selanjutnya aplikasi akan mengganti nama direktori “dirname” menjadi “newname”.

Selanjutnya perhatikan rutin rutin di bawah ini :

```

else:
    print("[ - ] directory {} not exists ... creating directory {}".format(dirname, dirname))
    _mkdir(dirname)
    if (os.path.isdir(newname) == False):
        _rename(dirname, newname)
    else:
        _rmdir(newname)
        _rename(dirname, newname)

```

Potongan kode di atas, akan dieksekusi jika direktori “dirname” tidak ditemukan, maka aplikasi akan membuat direktori “dirname” selanjutnya melakukan pengecekan apakah direktori “newname” sudah ada, jika belum ada maka aplikasi akan melakukan rename direktori “dirname” menjadi “newname”, Sebaliknya jika direktori “newname” sudah ada maka aplikasi akan menghapus direktori “newname” kemudian baru mengganti nama direktori “dirname” menjadi “newname”.

Pengecekan pengecekan di atas intinya berguna untuk menghindari raise exception pada aplikasi yang menyebabkan aplikasi crash.

2.5.2. Operasi File

Pada bagian ini kita akan membuat beberapa contoh aplikasi di python yang berhubungan dengan operasi file.

Mengecek Keberadaan File

Untuk mengecek apakah file ada atau tidak kita bisa menggunakan method `os.path.isfile`. Berikut ini adalah contoh, buat file baru dengan nama `isfile_sample.py` :

```
#!/usr/bin/env python3
"""
isfile_sample.py
check whether a file exists or not for course training
at www.jasaplus.com
"""
import os

if __name__ == "__main__":
    try:
        full_path = "filename.txt"
        if os.path.isfile(full_path):
            print("exists !")
        else:
            print("not found")
    except:
        raise
```

Penjelasan

```
if __name__ == "__main__":
```

Baris ini merupakan entry point untuk aplikasi `isfile_sample.py`.

```
    full_path = "filename.txt"
    if os.path.isfile(full_path):
        print("exists !")
```

Pada baris baris ini kita lakukan pengecekan apakah file dengan nama `filename.txt` ada, jika ada maka akan menampilkan pesan “exists !”

Membuat File Baru , Menulisi File, Membaca Isi File dan Menambah Isi File

Untuk membuat file baru kita bisa menggunakan fungsi `open()`.

Untuk menulisi atau menambah isi file, kita bisa menggunakan fungsi `write()` atau `writelines()`.

Untuk membaca isi file, kita bisa menggunakan fungsi `read()` atau `realine()` atau `readlines()`.

Berikut ini adalah contoh aplikasi untuk membuat file baru, menulisi file, membaca file dan menambah isi file.

Buat script baru dengan nama `file_sample.py`

```

#!/usr/bin/env python3
"""
file_sample.py
- create new file
- write file content
- read file
- append file content
- read file
material for course training
at www.jasaplus.com
"""

def create_write_file(file_path):
    try:
        print("[+] writing new file : " + file_path)
        with open(file_path, 'w') as fp:
            fp.write("write new file content\n")
    except:
        raise

def read_file(file_path):
    try:
        print("[+] reading file content : " + file_path)
        with open(file_path, 'r') as fp:
            data = fp.read()
            print("file content : \n")
            print(data)
    except:
        raise

def append_file(file_path):
    try:
        print("[+] appending file content : " + file_path)
        with open(file_path, 'a+') as fp:
            fp.write("\nappended text\n")
    except:
        raise

if __name__ == "__main__":
    file_path = "sample.txt"
    create_write_file(file_path)
    input("\npress any key to continue reading file")
    read_file(file_path)
    input("\npress any key to continue append file content")
    append_file(file_path)
    input("\npress any key to continue reading file")
    read_file(file_path)

```

Penjelasan

```

def create_write_file(file_path):
    try:
        print("[+] writing new file : " + file_path)
        with open(file_path, 'w') as fp:
            fp.write("write new file content\n")
    except:
        raise

```

fungsi `create_write_file`, digunakan untuk membuat file baru. Pada contoh di atas, kita membuat objek file descriptor dengan nama “fp” di mana kita gunakan `with` saat `open()` dengan modus `write` agar file descriptor secara otomatis ditutup setelah operasi `with` selesai. Setelah objek file descriptor “fp” siap, selanjutnya kita lakukan penulisan isi file dengan `write()`.

```
def read_file(file_path):
    try:
        print("[+] reading file content : " + file_path)
        with open(file_path, 'r') as fp:
            data = fp.read()
            print("file content : \n")
            print(data)
    except:
        raise
```

fungsi `read_file`, digunakan untuk membaca dan menampilkan isi file.

```
with open(file_path, 'r') as fp:
```

kita gunakan `with` agar file descriptor otomatis ditutup setelah operasi pada blok `with` selesai. Di sini kita `open()` dengan modus baca (`read`).

```
data = fp.read()
print("file content : \n")
print(data)
```

Data dari file descriptor “fp” selanjutnya disimpan ke variabel string “data” dengan method `read()`, selanjutnya data ditampilkan dengan fungsi built-in `print()`.

```
def append_file(file_path):
    try:
        print("[+] appending file content : " + file_path)
        with open(file_path, 'a+') as fp:
            fp.write("\nappended text\n")
    except:
        raise
```

Fungsi `append_file` digunakan untuk menambah isi file. Pada contoh ini kita siapkan file descriptor dengan modus penambahan (`append`) yaitu ‘a+’

Selanjutnya penulisan file dilakukan dengan method `write()`.

Menghapus File

Untuk menghapus file kita bisa menggunakan method `remove()`. Buat file baru dengan nama `file_del.py`

```
#!/usr/bin/env python3
'''
file_del.py
deleting file
material for course training
at www.jasaplus.com
```

```

'''
import os

if __name__ == "__main__":
    fname = input("\ntype file path that will be removed :").strip()
    if (len(fname) > 0):
        if os.path.isfile(fname):
            os.remove(fname)
            if os.path.isfile(fname):
                print("[-] failed to delete file")
            else:
                print("[+] file deleted successfully")
        else:
            print("file not exists !")
    else:
        print("[-] no file specified")

```

Penjelasan

```
import os
```

Pada baris ini kita mengimport modul os

```
fname = input("\ntype file path that will be removed :").strip()
```

Pada baris ini, kita simpan nama file berupa input dari user ke dalam variabel string fname. Fungsi strip() berguna untuk menghilangkan trailing space pada inputan.

```

if (len(fname) > 0):
    if os.path.isfile(fname):

```

Pada baris ini, kita lakukan validasi di mana kita akan menghapus file jika panjang input nama file > 0.

Selanjutnya kita lakukan pengecekan apakah file ada, jika file ada maka file akan dihapus dengan os.remove().

Kopi File

Untuk mengkopi file kita bisa menggunakan method dari shutil yaitu : copyfile(), copyfileobj() atau copy2(). Buat file baru dengan nama file_kopren.py

```

#!/usr/bin/env python3
'''
file_kopren.py
copy file and rename file sample
material for course training
at www.jasaplus.com
'''

import os
from shutil import copyfile, copyfileobj, copy2

def CopyFile(fname):
    try:
        print("[+] copy file using shutil.copyfile()")
        fname_copied = "copyfile_" + fname

```

```

        copyfile(fname, fname_copied)
        print("[+] copyfile() done !")
    except:
        raise

def CopyFileObj(fname):
    try:
        print("[+] copy file using shutil.copyfileobj()")
        fname_copied = "copyfileobj_" + fname
        bufsize = 16 * 1024
        with open(fname, 'rb') as src:
            with open(fname_copied, 'wb') as dest:
                copyfileobj(src, dest, bufsize)
        print("[+] copyfileobj() done !")
    except:
        raise

def Copy2(fname):
    try:
        print("[+] copy file using shutil.copy2()")
        fname_copied = "copy2_" + fname
        copy2(fname, fname_copied)
        print("[+] copy2() done !")
    except:
        raise

if __name__ == "__main__":
    fname = input("\ntype file name to copy:").strip()
    CopyFile(fname)
    CopyFileObj(fname)
    Copy2(fname)

```

Penjelasan

```

import os
from shutil import copyfile, copyfileobj, copy2

```

pada baris ini kita import modul os, dan kita impor fungsi copyfile, copyfileobj dan copy2. Perhatian jangan melakukan import * jika tidak terpaksa karena pada saat runtime akan menjadi boros memori karena fungsi fungsi yang tidak dibutuhkan juga terimport.

```

def CopyFile(fname):
    try:
        print("[+] copy file using shutil.copyfile()")
        fname_copied = "copyfile_" + fname
        copyfile(fname, fname_copied)
        print("[+] copyfile() done !")
    except:
        raise

```

CopyFile merupakan fungsi wrapper untuk shutil.copyfile(). Pada contoh ini kita tidak perlu menuliskan shutil.copyfile() karena copyfile() sudah diimport sebelumnya.

```

def CopyFileObj(fname):
    try:
        print("[+] copy file using shutil.copyfileobj()")
        fname_copied = "copyfileobj_" + fname
        bufsize = 16 * 1024
        with open(fname, 'rb') as src:

```



```

        with open(fname_copied, 'wb') as dest:
            copyfileobj(src, dest, bufsize)
        print("[+] copyfileobj() done !")
    except:
        raise

```

CopyFileObj() merupakan fungsi wrapper yang kita buat untuk menjalankan copyfileobj(). Pertama tama kita baca dulu isi file sebagai binary

Perhatikan !

```

with open(fname, 'rb') as src:

```

Dengan menggunakan with, maka secara otomatis file descriptor “src” akan ditutup setelah penggunaan selesai, di sini kita baca isi file name yang akan dikopi lalu disimpan di objek file descriptor dengan nama “src”. Pada baris ini kita menyiapkan modus untuk file descriptor berupa read binary.

```

        with open(fname_copied, 'wb') as dest:

```

Selanjutnya kita akan menyiapkan file descriptor untuk target penulisan isi file descriptor src tadi di mana kita gunakan nama file descriptor “dest” dan nama file fname_copied. Pada baris di atas, menggunakan parameter ‘wb’ artinya kita akan menyiapkan data file descriptor dalam mode write binary.

Setelah semua siap, kita lakukan copyfileobj sebesar bufsize , di mana bufsize merupakan ukuran buffer, pada contoh ini kita gunakan ukuran buffer 16 x 1024 bytes, untuk file berukuran kecil ukuran buffer ini mencukupi, untuk file berukuran besar, agar pengkopian data bisa lebih cepat sebaiknya kita gunakan ukuran buffer yang lebih besar.

```

            copyfileobj(src, dest, bufsize)

```

Selanjutnya :

```

def Copy2(fname):
    try:
        print("[+] copy file using shutil.copy2()")
        fname_copied = "copy2_" + fname
        copy2(fname, fname_copied)
        print("[+] copy2() done !")
    except:
        raise

```

Fungsi Copy2() merupakan fungsi wrapper yang kita buat untuk menjalankan fungsi copy2(). Fungsi copy2 memiliki fitur lebih lengkap dari copy() di mana pada copy2() ditambahkan pengkopian metadata file yang dikopi untuk file baru.

2.6. OOP PADA PYTHON

Python mendukung bahasa pemrograman berorientasi objek (OOP). OOP (object oriented programming) merupakan konsep / paradigma pemrograman yang berorientasi pada objek . Tujuan dari OOP adalah untuk mempermudah pengembangan program dengan cara mengikuti model yang telah ada di kehidupan sehari-hari. Suatu aplikasi yang menerapkan konsep OOP bisa memiliki 4 karakteristik dasar OOPS : abstraction, inheritance, encapsulation dan polymorphism.

2.6.1. Class, Object dan Method

class merupakan suatu model / cetak biru / kerangka dasar yang berisi kumpulan atribut, metode dalam satu unit.

Sebagai contoh analogi : family Felidae atau kucing merupakan kelas yang mendefinisikan family kucing berupa karakteristik yang dimiliki secara garis besar dari objek objek di dalamnya.

Pada saat pembentukan instance dari suatu class, maka ada beberapa magic method yang secara otomatis dipanggil :

1. `__new__`

magic method ini akan dipanggil saat proses konstruksi suatu instance (constructor)

2. `__init__`

magic method ini akan dipanggil setelah object instance terbentuk (initializer)

3. `__call__`

magic method ini akan dipanggil setelah `__init__`

object merupakan perwujudan (turunan) dari class, setiap object akan mempunyai attribute dan method yang dimiliki oleh class-nya.

Contoh sederhana untuk menjelaskan object, contoh : kelas kucing memiliki objek : angora, kucing kampung, harimau, leopard dan lain sebagainya yang termasuk ke dalam kelas “kucing”

Untuk contoh class, buat file baru dengan nama kelas.py :

```
#!/usr/bin/env python3
"""
kelas.py
sample class for python3 full stack training
at www.jasaplus.com
"""
class kucing:
    sound = "?"
    habit = "?"

    def __init__(self):
        self.sound = "meow"
        self.habit = "pee"

    def DoHabit(self):
        print("doing " + self.habit)
```

```
def DoSound(self):  
    print(self.sound)
```

Penjelasan

class kucing:

Baris di atas digunakan untuk membuat class baru dengan nama identifier : kucing.

```
sound = "?"  
habit = "?"
```

Merupakan class variable yang didefinisikan di dalam class kucing.

```
def __init__(self):
```

Method `__init__` merupakan initializer yang secara otomatis akan dijalankan saat proses Instantiation (pembuatan objek dari class), tepatnya setelah objek baru diciptakan.

```
    self.sound = "meow"  
    self.habit = "pee"
```

Pada method initializer, kita mengisi variabel class dengan string (method initializer merupakan magic method yang akan dipanggil pertama kali setelah objek instance baru saja dibentuk)

```
def DoHabit(self):  
    print("doing " + self.habit)  
  
def DoSound(self):  
    print(self.sound)
```

Baris baris ini merupakan method yang dimiliki class kucing.

Selanjutnya, untuk contoh object, buat file python baru dengan nama objek.py :

```
#!/usr/bin/env python3  
  
"""  
objek.py  
sample object for python3 full stack training  
at www.jasaplus.com  
"""  
  
from kelas import kucing  
  
anggora = kucing()  
anggora.DoHabit()  
anggora.DoSound()
```

Simpan file objek.py di direktori yang sama dengan kelas.py.

Untuk mencoba jalanya script di atas, jalankan file objek.py :

```
ringlayer@ringlayer-Inspiron-3442:~/Desktop/TRAINING/PYTHON/source/oop$ python3 objek.py
doing pee
meow
ringlayer@ringlayer-Inspiron-3442:~/Desktop/TRAINING/PYTHON/source/oop$
```

Penjelasan

```
from kelas import kucing
```

Baris ini digunakan untuk mengimport class kucing.

```
anggora = kucing()
```

Baris ini merupakan proses Instantiation (pembuatan objek dari class), di mana kita membuat objek baru dengan nama identifier anggora yang merupakan instance object dari class kucing.

```
anggora.DoHabit()
anggora.DoSound()
```

Baris di atas digunakan untuk menjalankan method dari class kucing yaitu DoHabit dan DoSound.

Method

Method merupakan blok kode yang berada di dalam suatu identifier nama, di mana blok kode tersebut biasanya digunakan untuk suatu tugas tertentu. Method merupakan fungsi yang dimiliki suatu class pada python, jadi yang membedakan antara fungsi dan method adalah method merupakan milik suatu kelas sedangkan fungsi adalah independen tidak dimiliki oleh suatu class.

Kembali lagi pada contoh script python sebelumnya kelas.py :

```
#!/usr/bin/env python3
"""
kelas.py
sample class for python3 full stack training
at www.jasaplus.com
"""
class kucing:
    sound = "?"
    habit = "?"

    def __init__(self):
        self.sound = "meow"
        self.habit = "pee"

    def DoHabit(self):
        print("doing " + self.habit)

    def DoSound(self):
        print(self.sound)
```

`__init__` , DoHabit dan DoSound merupakan method (fungsi yang dimiliki class) pada class kucing.

2.6.2. Instantiation

Instantiation adalah istilah yang digunakan untuk proses pembuatan instance object dari suatu class.

Pada contoh di atas, instantiation kita lakukan pada rutin ini :

```
anggora = kucing()
```

Baris di atas akan membuat instance object baru dari class kucing dengan nama identifier :
“anggora”

2.6.3. Inner Class

inner class merupakan class yang berada dalam suatu class.

Untuk contoh inner class, buat file python baru dengan nama inner_class.py :

```
#!/usr/bin/env python3

class Robot():
    sound = "?"
    steering = "?"
    name = "?"

    def __init__(self):
        self.sound = "beep"
        self.name = "wall e"

    def Steering(self):
        self.steering = "skid steering"

    def DoIdle(self):
        print("[+] do nothing")

    def MyProperties(self):
        print("[+] sound : " + self.sound)
        print("[+] steering : " + self.steering)
        print("[+] name : " + self.name)

class Head:
    def _turn_left():
        print("[+] robot head turn left")
    def _turn_right():
        print("[+] robot head turn right")

class Arm:
    def _pick_object():
        print("[+] robot arm pick object")
    def _push_object():
        print("[+] robot arm push object")

if __name__ == "__main__":
    print("[+] start")
    wall_e = Robot()
    wall_e.Steering()
    wall_e.MyProperties()
```

```
wall_e.DoIdle()
wall_e.Head._turn_left()
wall_e.Head._turn_right()
wall_e.Arm._pick_object()
wall_e.Arm._push_object()
```

```
ringlayer@ringlayer-Inspiron-3442:~/Desktop/TRAINING/PYTHON/source/oop$ python3 inner_class.py
[+] start
[+] sound : beep
[+] steering : skid steering
[+] name : wall e
[+] do nothing
[+] robot head turn left
[+] robot head turn right
[+] robot arm pick object
[+] robot arm push object
ringlayer@ringlayer-Inspiron-3442:~/Desktop/TRAINING/PYTHON/source/oop$
```

Penjelasan

class Robot():

Digunakan membuat class baru dengan nama identifier : “Robot”

```
sound = "?"
steering = "?"
name = "?"
```

Baris baris di atas digunakan untuk mendefinisikan beberapa class variable (variable yang dimiliki class Robot).

```
def __init__(self):
    self.sound = "beep"
    self.name = "wall e"
```

`__init__` merupakan method initializer untuk class Robot. Method ini akan dijalankan saat proses instantiation (pembuatan object instance dari class).

```
def Steering(self):
    self.steering = "skid steering"

def DoIdle(self):
    print("[+] do nothing")

def MyProperties(self):
    print("[+] sound : " + self.sound)
    print("[+] steering : " + self.steering)
    print("[+] name : " + self.name)
```

Steering, DoIdle dan MyProperties merupakan method yang dimiliki oleh class Robot, di mana method ini masing masing memiliki 1 parameter argumen self. Self merupakan variable yang merupakan representasi instance dari class tersebut.

2.6.4. Class Variable dan Instance Variable

Class variable dideklarasikan dalam suatu class, class variable memiliki nilai yang sama saat instantiation masing-masing objek dari class tersebut, sedangkan instance variable merupakan variabel yang nilainya berbeda untuk masing-masing objek hasil instantiation.

Untuk lebih jelasnya berikut ini contoh class variable :

```
#!/usr/bin/env python3

class Hacker():
    skills = "advanced"
    capabilities = "outstanding"
    programming_skills = "great"

    def Hello(self):
        print("[+] simply saying hello")
```

Penjelasan

Pada contoh di atas, variabel `skills`, `capabilities` dan `programming_skills` adalah class variable (variabel yang dimiliki class).

Berikut ini contoh instance variable :

```
#!/usr/bin/env python3

from class_variable import Hacker
ringlayer = Hacker()
ringlayer.Hello()
ringlayer.skills = "hacking"
print("[+] Ringlayer skills : {}".format(ringlayer.skills))
```

Penjelasan

Pada contoh di atas, pada baris :

```
ringlayer.skills = "hacking"
```

di sini kita membuat “instance variable” khusus milik instance objek `ringlayer`.

2.6.5. Inheritance

Inheritance merupakan proses mewariskan karakteristik dari suatu kelas untuk kelas turunanya. Untuk mewariskan metode dan atribut dari kelas parent maka saat deklarasi child kelas,

Buat file python baru dengan nama inheritance.py :

```
#!/usr/bin/env python3
"""
created by Antonius Ringlayer
material for python3 course
"""
#parent class :
class Robot():
    sound = "?"
    steering = "?"
    name = "?"

    def __init__(self):
        self.sound = "beep"
        self.name = "wall e"

    def Steering(self):
        self.steering = "skid steering"

    def DoIdle(self):
        print("[+] do nothing")

    def MyProperties(self):
        print("[+] sound : " + self.sound)
        print("[+] steering : " + self.steering)
        print("[+] name : " + self.name)

class Head:
    def _turn_left():
        print("[+] robot head turn left")
    def _turn_right():
        print("[+] robot head turn right")

class Arm:
    def _pick_object():
        print("[+] robot arm pick object")
    def _push_object():
        print("[+] robot arm push object")

#inheritance class :
class ThreeWheeledRobot(Robot):
    def __init__(self):
        self.sound = "squeeze"
        self.name = "differential drive robot"
        self.steering = "differential drive"

    #method override
    def DoIdle(self):
        print("[+] idle override")

    def AdditionalMethod(self):
        print("[+] Hello I have new method")

#instantiation
```



```
bot = ThreeWheeledRobot()
bot.MyProperties()
bot.DoIdle()
bot.AdditionalMethod()
bot.Head._turn_left()
bot.Arm._pick_object()
```

Penjelasan

Pada contoh di atas, kita memiliki parent class : Robot

```
class Robot():
```

Masih di script yang sama (inheritance.py), kita membuat child class yang merupakan turunan / inheritance dari class Robot :

```
#inheritance class :
class ThreeWheeledRobot(Robot):
```

Di contoh ini kita membuat class baru dengan nama ThreeWheeledRobot, di mana class ini akan menginherit / menurunkan karakteristik (method, variabel) dari parent class.

```
def __init__(self):
    self.sound = "squeeze"
    self.name = "differential drive robot"
    self.steering = "differential drive"

#method override
def DoIdle(self):
    print("[+] idle override")

def AdditionalMethod(self):
    print("[+] Hello I have new method")
```

Class ThreeWheeledRobot memiliki 2 method bawaan dari class parent yaitu : `__init__` dan `DoIdle`. Di mana pada class ini memiliki implementasi yang berbeda dari class parent. Method override akan mengesampingkan method asli bawaan dari parent class, di mana implementasi method pada class ThreeWheeledRobot memiliki implementasi sendiri pada blok method `__init__` dan `DoIdle`.

AdditionalMethod adalah method tambahan yang dimiliki class ThreeWheeledRobot, method ini tidak ada pada parent class.

Untuk menguji jalanya method method pada class kita melakukan instantiation :

```
#instantiation

bot = ThreeWheeledRobot()
bot.MyProperties()
bot.DoIdle()
bot.AdditionalMethod()
bot.Head._turn_left()
bot.Arm._pick_object()
```

Berikut ini hasil dari menjalankan script di atas :

```
ringlayer@ringlayer-Inspiron-3442: ~/Desktop/TRAINING/PYTHON/source/oop
File Edit Tabs Help
root@ringlay... x ringlayer@rin... x
ringlayer@ringlayer-Inspiron-3442:~/Desktop/TRAINING/PYTHON/source/oop$ ./inheritance.py
[+] sound : squeeze
[+] steering : differential drive
[+] name : differential drive robot
[+] idle override
[+] Hello I have new method
[+] robot head turn left
[+] robot arm pick object
ringlayer@ringlayer-Inspiron-3442:~/Desktop/TRAINING/PYTHON/source/oop$
```

2.6.6. Encapsulation

Encapsulation merupakan pembungkusan atribut, metode ke dalam satu unit tunggal dan membatasi akses pada beberapa komponen pada objek.



Sebagai analogi, suatu kapsul merupakan satu kesatuan yang terdiri dari berbagai macam jenis obat di dalamnya. Pada enkapsulasi, data di dalam suatu kelas tersembunyi di dalam kelas lainnya.

Enkapsulasi bertujuan membatasi akses ke atribut (variabel, method) di dalam class secara tidak disengaja.

Untuk mewujudkan encapsulation bisa dilakukan dengan membuat akses modifier private untuk variabel dan method pada suatu class yaitu dengan menambahkan tanda __ (underscore 2 kali), selain itu kita juga membutuhkan initializer __init__ di dalam class tersebut untuk melakukan inisialisasi nilai variabel private yang disembunyikan dari luar class.

Initializer ini akan dipanggil pertama kali saat objek instance dari class telah terbentuk.

Bagaimana mengambil dan mengubah nilai variabel private pada suatu class ?
Caranya adalah dengan membuat suatu method getter dan setter (sama seperti pada java).

Lalu bagaimana cara memanggil method private pada suatu class ?
Dengan pola penulisan calling convention di bawah ini :

namaobjekinstance._namaclass__methodprivate()

Untuk lebih jelasnya buat contoh file baru dengan nama encapsultion.py :

```

#!/usr/bin/env python3
"""
python encapsulation example
made by antonius ringlayer
for python3 training material
at www.jasaplus.com
"""
class Capsule:
    def __init__(self):
        self.data1 = "data 1"
        self.__data2 = "data 2"

    def DoItinMyOwnClass(self):
        print("data1 : " + self.data1)
        print("__data2 : " + self.__data2)
        self.__PrivateMethod()

    def GetterMethod(self):
        return self.data1, self.__data2

    def SetterMethod(self, val1, val2):
        self.data1 = val1
        self.__data2 = val2

    def PublicMethod(self):
        print("I am public method")

    def __PrivateMethod(self):
        print("This is a private method")

capsul = Capsule()
capsul.PublicMethod()
capsul._Capsule__PrivateMethod()

#getter method
get1,get2 = capsul.GetterMethod()
print("data1 from getter method : {}".format(get1))
print("data2 from getter method : {}".format(get2))

#setter method
capsul.SetterMethod("changed 1", "changed 2")
get1,get2 = capsul.GetterMethod()
print("data1 from getter method : {}".format(get1))
print("data2 from getter method : {}".format(get2))

#we can access private variable and call private method within class
#here is the demo :
capsul.DoItinMyOwnClass()

```

Jalankan script di atas, hasilnya :

```

ringlayer@ringlayer-Inspiron-3442:~/Desktop/TRAINING/PYTHON/source/oop$ ./encapsulation.py
I am public method
This is a private method
data1 from getter method : data 1
data2 from getter method : data 2
data1 from getter method : changed 1
data2 from getter method : changed 2
data1 : changed 1
data2 : changed 2
This is a private method

```

Penjelasan

```
class Capsule:
    def __init__(self):
        self.data1 = "data 1"
        self.__data2 = "data 2"
```

Pada baris kode di atas, kita membuat class baru dengan nama identifier Capsule. Class capsule menggunakan initializer `__init__` untuk menginisialisasi nilai variabel public `data1` dan `__data2`.

Variabel `data1` adalah variabel public karena tidak menggunakan access modifier `__` (double under score). Variabel ini bisa langsung diakses oleh instance object dari class Capsule.

Variabel `__data2` adalah variabel private, variabel ini menggunakan modifier akses private berupa `__` (double under score).

```
def DoItinMyOwnClass(self):
    print("data1 : " + self.data1)
    print("__data2 : " + self.__data2)
    self.__PrivateMethod()
```

Method `DoItinMyOwnClass` mendemonstrasikan bahwa variabel private bisa diakses langsung selama scope run time masih berada di dalam class tersebut. Selain itu method ini juga bisa digunakan untuk memanggil method private `__PrivateMethod()`.

Sehingga dari instance object capsul, kita bisa menampilkan variabel private `__data2` dan method private `__PrivateMethod()` dengan cara memanggil method `DoItinMyOwnClass()`. Contoh :

```
capsul.DoItinMyOwnClass
```

Sebaliknya kita tidak bisa memanggil secara langsung method `__PrivateMethod` dan tidak bisa secara langsung menampilkan nilai variabel `__data2`. Contoh kode di bawah ini adalah salah dan akan menghasilkan error karena kita berusaha mengakses `__PrivateMethod` dan `__data2` yang secara langsung dari objek instance

```
capsul = Capsule()
capsul.__PrivateMethod() → Error karena private method tidak bisa dipanggil secara langsung dari instance object.
```

```
print(capsul.__data2) → Error karena kita tidak bisa secara langsung mengakses private method dari instance object.
```

Kembali lagi ke kode `encapsulation.py` di atas, kita memiliki fungsi getter dan setter :

```
def GetterMethod(self):
    return self.data1, self.__data2
```

Merupakan fungsi getter yang berguna untuk mengambil nilai dari variabel public dan private di class ke dalam bentuk list (variabel public sebenarnya bisa langsung diakses tanpa perlu method getter)

```
def SetterMethod(self, val1, val2):  
    self.data1 = val1  
    self.__data2 = val2
```

Merupakan method setter yang berguna untuk mengubah nilai dari variabel private dan public di dalam class (variabel public sebenarnya bisa langsung diubah tanpa perlu setter method).

```
def PublicMethod(self):  
    print("I am public method")  
  
def __PrivateMethod(self):  
    print("This is a private method")
```

PublicMethod merupakan method public yang bisa dipanggil langsung dari instance object. __PrivateMethod merupakan method private yang tidak bisa secara langsung dipanggil dari instance object, kecuali dengan calling convention dengan pola :
namaobjekinstance._namakelas__namaprivatemethod()

Berikut ini contoh memanggil public method dan private method :

```
capsul = Capsule()  
capsul.PublicMethod()  
capsul._Capsule__PrivateMethod()
```

Berikut ini contoh mengambil nilai variabel private ke dalam list yang telah disiapkan dengan method getter :

```
#getter method  
get1,get2 = capsul.GetterMethod()  
print("data1 from getter method : {}".format(get1))  
print("data2 from getter method : {}".format(get2))
```

Berikut ini contoh mengubah nilai dengan method setter :

```
#setter method  
capsul.SetterMethod("changed 1", "changed 2")  
get1,get2 = capsul.GetterMethod()  
print("data1 from getter method : {}".format(get1))  
print("data2 from getter method : {}".format(get2))
```

2.6.7. Method Overriding dan Method OverLoading

Method Overriding

Method overriding merupakan kemampuan dari suatu child class untuk mengoverride (mengesampingkan / mengubah / menimpa) implementasi dari suatu method milik parent class di mana child class akan memiliki identifier nama method yang sama tetapi memiliki implementasi berbeda untuk suatu method.

Untuk lebih jelasnya, buat contoh file baru dengan nama method_overriding.py :

```
#!/usr/bin/env python3
#sample method overriding
#made by Antonius Ringlayer
#www.jasaplus.com - www.ringlayer.com
#prepare new class
class Kelas:
    def Method1(self):
        print("original method code block")

#inheritance
class KelasDescendant(Kelas):
    #overriding
    def Method1(self):
        print("Method overriding")

kelas2 = KelasDescendant()
kelas2.Method1()
```

Penjelasan

Pada bagian pertama kita memiliki parent class dengan identifier nama “Kelas”

```
class Kelas:
    def Method1(self):
        print("original method code block")
```

Selanjutnya kita lakukan inheritance dari class parent di mana kita melakukan override pada Method1 :

```
#inheritance
class KelasDescendant(Kelas):
    #overriding
    def Method1(self):
        print("Method overriding")
```

Kita menguji hasil method override dengan membuat instance object baru dari kelas KelasDescendant :

```
kelas2 = KelasDescendant()
kelas2.Method1()
```

Jalankan script di atas, maka hasilnya :

```
ringlayer@ringlayer-Inspiron-3442:~/Desktop/TRAINING/PYTHON/source/oop$ ./method_overriding.py
Method overriding
ringlayer@ringlayer-Inspiron-3442:~/Desktop/TRAINING/PYTHON/source/oop$
```

Method Overloading

Method overloading merupakan cara pemanggilan suatu method dengan cara berbeda, di mana kita bisa menambahkan parameter pada suatu method. Pada dasarnya method overloading tidak disupport di python. Tetapi kita bisa menggunakan trik dengan menyupply default value untuk parameter sehingga saat tidak ada parameter yang diberikan maka secara default method akan menganggap parameter menggunakan nilai default.

Untuk lebih jelasnya, buat contoh file baru dengan nama method_overloading.py :

```
#!/usr/bin/env python3
"""
sample method overloading
made by Antonius Ringlayer
www.jasaplus.com - www.ringlayer.com

by default method overloading is not supported in python.
We use default value for parameter to create a method that can be
called without parameter or with parameter
since we have supplied default parameter value when
there's no parameter supplied
"""

#prepare new class
class MoDemo():
    def MyPublicMethod(self, data=None):
        if data != None:
            print("[+] data : {}".format(data))
        else:
            print("[+] do nothing")

obj = MoDemo()
obj.MyPublicMethod()
#call it different way :
obj.MyPublicMethod("hack")
```

Penjelasan

Dari instance object obj kita bisa memanggil method MyPublicMethod dengan 2 cara yang berbeda yaitu dengan menggunakan parameter maupun tidak menggunakan parameter :

```
obj.MyPublicMethod()  
#call it different way :  
obj.MyPublicMethod("hack")
```

Perbedaan Method Overriding dan Method Overloading

Pada python method overriding bertujuan untuk mengesampingkan implementasi method yang sebelumnya dengan implementasi baru (override) sedangkan method overloading hanyalah pemanggilan suatu method dengan cara yang berbeda di mana parameter dari suatu method bisa ditambah (ekstensi dari suatu method)

2.6.8. Polymorphism

Polymorphism berasal dari 2 kata : “poly” yang artinya banyak dan kata “morphism” yang artinya bentuk. Jadi polymorphism bisa diartikan sebagai banyak bentuk.

Secara sederhana polymorphism adalah konsep di mana suatu aplikasi bisa mengeksekusi satu aksi dengan berbagai cara yang berbeda.

Untuk mengimplementasikan polymorphism pada oop di python, kita bisa membuat suatu class parent di mana class tersebut memiliki child class yang memiliki implementasi berbeda beda untuk method method bawaan dari class parent.

Untuk lebih jelasnya, buat script python baru dengan nama polymorphism.py

```
#!/usr/bin/env python3

'''
polymorphism sample
created by Antonius Ringlayer
www.jasaplus.com - www.ringlayer.com

oop material for python3 course
'''

class Robot():
    def __init__(self, name):
        self.name = name

    def Steering(self):
        print("[+] By default, my steering is differential drive")

    def Brain(self):
        print("[+] By default, my brain consist of deep learning models")

    def SayMyName(self):
        print("My name is {}".format(self.name))

class WallE(Robot):
    def __init__(self, name):
        self.name = name

    def Steering(self):
        print("[+] {} : By default, my steering is modern skid steering".format(self.name))

    def Brain(self):
        print("[+] {} : By default, my brain consist of digital circuit".format(self.name))

    def SayMyName(self):
        print("[+] My name is {}".format(self.name))

class Johnny5(Robot):
    def __init__(self, name):
        self.name = name

    def Steering(self):
        print("[+] {} : By default, my steering is classic skid steering".format(self.name))
```

```

def Brain(self):
    print("[+] {} : By default, my brain consist of analog circuit".format(self.name))

def SayMyName(self):
    print("[+] My name is {}".format(self.name))

walle = WallE('wall-e')
johnny5 = Johnny5('johnny5')

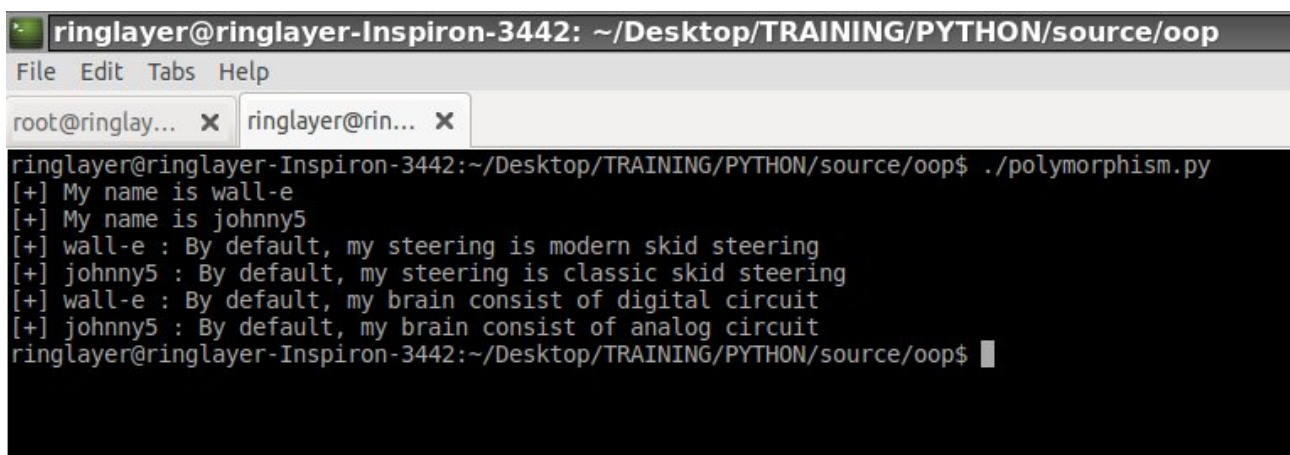
walle.SayMyName()
johnny5.SayMyName()

walle.Steering()
johnny5.Steering()

walle.Brain()
johnny5.Brain()

```

Simpan script di atas dengan nama polymorphism.py lalu jalankan :



```

ringlayer@ringlayer-Inspiron-3442: ~/Desktop/TRAINING/PYTHON/source/oop
File Edit Tabs Help
root@ringlay... x ringlayer@rin... x
ringlayer@ringlayer-Inspiron-3442:~/Desktop/TRAINING/PYTHON/source/oop$ ./polymorphism.py
[+] My name is wall-e
[+] My name is johnny5
[+] wall-e : By default, my steering is modern skid steering
[+] johnny5 : By default, my steering is classic skid steering
[+] wall-e : By default, my brain consist of digital circuit
[+] johnny5 : By default, my brain consist of analog circuit
ringlayer@ringlayer-Inspiron-3442:~/Desktop/TRAINING/PYTHON/source/oop$

```

Penjelasan

```

class Robot():
    def __init__(self, name):
        self.name = name

    def Steering(self):
        print("[+] By default, my steering is differential drive")

    def Brain(self):
        print("[+] By default, my brain consist of deep learning models")

    def SayMyName(self):
        print("My name is {}".format(self.name))

```

Pada contoh di atas, kita memiliki class parent dengan nama Robot, di mana class ini memiliki satu initializer (`__init__`) dan method Steering, Brain dan SayMyName.

Initializer akan dipanggil pertama kali setelah objek instance terbentuk.

Implementasi polymorphism diwujudkan dengan membuat implementasi yang berbeda untuk method Steering dan method Brain pada masing masing class child (inheritance dari parent class)

Pada class turunan dengan nama identifier Walle :

```
def Steering(self):  
    print("[+] {} : By default, my steering is modern skid steering".format(self.name))  
  
def Brain(self):  
    print("[+] {} : By default, my brain consist of digital circuit".format(self.name))
```

Sedangkan pada class turunan dengan nama identifier Johnny5 memiliki implementasi yang berbeda untuk method Steering dan method Brain:

```
def Steering(self):  
    print("[+] {} : By default, my steering is classic skid steering".format(self.name))  
  
def Brain(self):  
    print("[+] {} : By default, my brain consist of analog circuit".format(self.name))
```

2.6.9. Abstraction

Abstraksi adalah proses pengabstraksian atau menyembunyikan kerumitan dari suatu proses (dalam hal ini program). Dengan abstraksi maka kerumitan implementasi bisa disembunyikan dari user.

Perbedaan antara abstraction dan encapsulation adalah tujuan dari kedua konsep tersebut. Abstraction bertujuan untuk menyembunyikan kerumitan proses sedangkan encapsulation bertujuan untuk membatasi akses ke atribut (variabel atau method) di dalam class.

Untuk implementasi konsep abstraction di python, bisa dilakukan dengan membuat abstract class. Suatu kelas abstract bisa memiliki metode abstract dan concrete method. Metode abstract tidak memiliki body dan implementasi sedangkan metode concrete memiliki body dan implementasi.

Kita tidak membuat instantiation untuk abstract class, suatu abstract class ditujukan sebagai blueprint untuk child class (inheritance dari abstract class).

Untuk membuat class abstract di python, kita bisa menggunakan modul ABC (abstract base classes). Setelah mengimport modul ABC, kita bisa menandai method yang ingin dijadikan abstract di dalam class dengan keyword `@abstractmethod`

Berikut ini contoh teknik programming dengan menerapkan abstraction dengan abstract class. Buat script python baru dengan nama abstraction.py

```
#!/usr/bin/env python3
"""
sample abstract class
for python3 training
at www.jasaplus.com
"""
from abc import ABC, abstractmethod
class Robot(ABC):
    @abstractmethod
    def Steering(self):
        pass

    @abstractmethod
    def GenericOperation(self):
        pass

#inheritance 1
class WallE(Robot):
    def Steering(self):
        print("[+] skid steering")

    def GenericOperation(self):
        print("[+] remove trash")

#inheritance 2
class Jibo(Robot):
    def Steering(self):
        print("[+] no steering")

    def GenericOperation(self):
        print("[+] joking")
```

```
#instantiations
print("=JIBO BORN=")
jibo = Jibo()
jibo.Steering()
jibo.GenericOperation()

print("\n=WALL-E BORN=")
walle = Walle()
walle.Steering()
walle.GenericOperation()
```

2.6.10. Penggunaan Operator Overloading pada Class

Operator overloading memungkinkan kita menggunakan operator matematika, logika dan bitwise dengan cara yang berbeda.

Untuk menerapkan operator overloading, python menyediakan beberapa magic method, dengan magic method memungkinkan kita melakukan operator overloading.

Berikut ini operator yang bisa kita overloading dan magic methodnya :

Operator	Magic Method
+	<code>__add__(self, other)</code>
-	<code>__sub__(self, other)</code>
*	<code>__mul__(self, other)</code>
/	<code>__truediv__(self, other)</code>
//	<code>__floordiv__(self, other)</code>
%	<code>__mod__(self, other)</code>
**	<code>__pow__(self, other)</code>
&	<code>__and__(self, other)</code>
	<code>__or__(self, other)</code>
^	<code>__xor__(self, other)</code>
>	<code>__gt__(self, other)</code>
>=	<code>__ge__(self, other)</code>
<	<code>__lt__(self, other)</code>
<=	<code>__le__(self, other)</code>
==	<code>__eq__(self, other)</code>
!=	<code>__ne__(self, other)</code>

Untuk lebih jelasnya, buat file baru dengan nama `operator_overloading.py` :

```
#!/usr/bin/env python3
"""
operator_overloading.py
sample operator overloading in class
python training material for jasaplus.com
"""
class Operation:
    def __init__(self, val):
        self.val = val

    def __add__(self, obj):
        return (self.val + obj.val - 1)

    def __sub__(self, obj):
        return self.val - obj.val
```

```

res1 = Operation(10)
res2 = Operation(5)

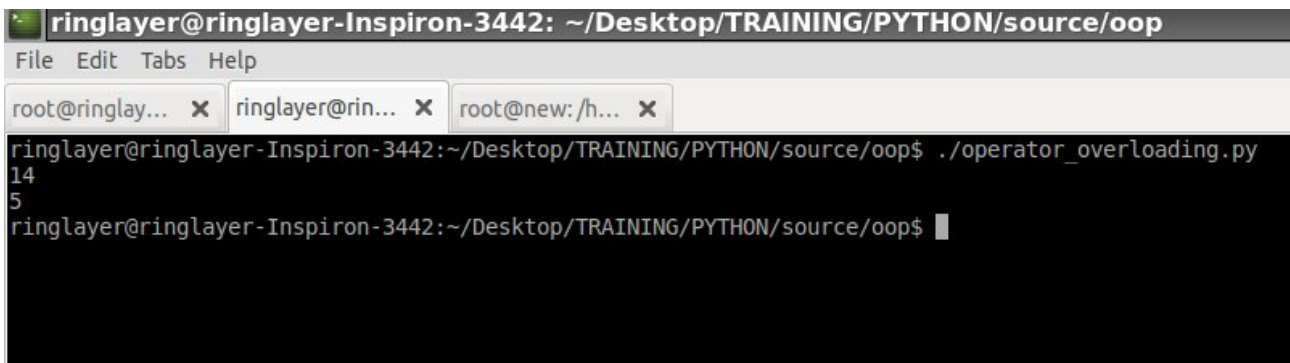
print(res1 + res2)

res3 = Operation(10)
res4 = Operation(5)

print(res1 - res2)

```

Simpan file di atas dengan nama operator_overloading.py lalu jalankan, hasilnya :



```

ringlayer@ringlayer-Inspiron-3442: ~/Desktop/TRAINING/PYTHON/source/oop
File Edit Tabs Help
root@ringlay... x ringlayer@rin... x root@new:/h... x
ringlayer@ringlayer-Inspiron-3442:~/Desktop/TRAINING/PYTHON/source/oop$ ./operator_overloading.py
14
5
ringlayer@ringlayer-Inspiron-3442:~/Desktop/TRAINING/PYTHON/source/oop$

```

Penjelasan

```

def __add__(self, obj):
    return (self.val + obj.val - 1)

```

Di sini kita menyiapkan magic method `__add__` untuk operator `+` yang akan kita gunakan.

```

res1 = Operation(10)
res2 = Operation(5)

print(res1 + res2)

```

Pada baris baris ini terlihat hasil dari operator overloading di mana hasilnya adalah : “14”, yang didapat dari nilai kembali magic method `__add__` yaitu : $10 + 5 - 1$

```

def __sub__(self, obj):
    return self.val - obj.val

```

Di sini kita siapkan magic method untuk operator : “-”

```

res3 = Operation(10)
res4 = Operation(5)
print(res1 - res2)

```

Hasil dari nilai kembali magic method akan menyebabkan operator “-” menghasilkan nilai 5 yang didapat dari $10 - 5$

2.7. CLOSURE, METAPROGRAMMING, DECORATOR DAN METACLASS

Pada bagian ini, kita akan mempelajari beberapa teknik pemrograman python yang menarik, yaitu : closure, decorator, metaprogramming dan metaclass.

Untuk bisa memahami teknik teknik seperti closure, decorator dan metaclass; membutuhkan pemahaman bahwa fungsi, method, variabel, sequence, class di python pada dasarnya merupakan objek. Misal class merupakan objek inheritance dari suatu metaclass, fungsi dan method juga merupakan objek.

Closure

Closure merupakan objek berupa fungsi yang berasal dari inner function (fungsi yang berada di dalam suatu fungsi) di mana objek ini mengingat nilai / menyertakan referensi nilai variabel yang berasal dari fungsi pembungkusnya.

Pada dasarnya tanpa closure, kita tidak bisa memanggil fungsi yang berada di dalam fungsi dari luar fungsi pembungkusnya (outer function). Dengan teknik closure memungkinkan kita memanggil fungsi yang berada di dalam fungsi (inner function) dari luar fungsi pembungkusnya (outer function).

Ciri dari closure :

- Kita harus memiliki inner function
- inner function harus merujuk ke variabel dari outer function.
- Outer function harus mereturn nilai kembali berupa objek yang merupakan inner function.

Tujuan Closure :

closure digunakan untuk menyembunyikan data dan menghindari penggunaan variabel global.

Untuk lebih jelasnya, buat contoh file dengan nama closure_sample.py

```
#!/usr/bin/env python3
"""
sample python closure for python training
at www.jasaplus.com
"""
def func1(a):
    def func2():
        print(a)
    return func2
obj = func1("remember me")
obj()
```

Penjelasan

```
def func1(a):
```

Kita memiliki fungsi pembungkus dengan nama func1.


```
def func2():  
    print(a)
```

func2 merupakan fungsi di dalam func1. Fungsi ini menampilkan variabel “a” dari outer function : func1.

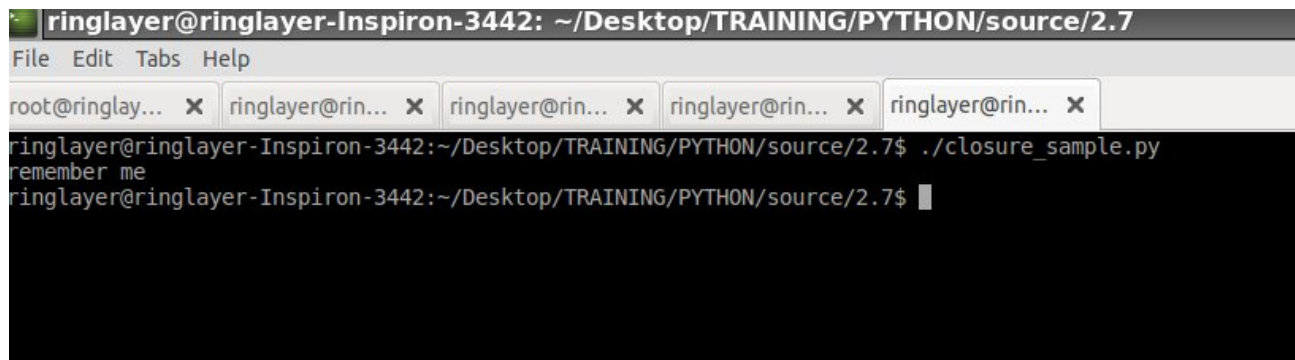
Pada func1 kita memiliki nilai kembali berupa objek yang merupakan fungsi “func2”:

```
    return func2
```

Nilai kembali disimpan pada variabel obj :

```
obj = func1("remember me")  
obj()
```

Hasil dari menjalankan script di atas :



```
ringlayer@ringlayer-Inspiron-3442: ~/Desktop/TRAINING/PYTHON/source/2.7  
File Edit Tabs Help  
root@ringlay... X ringlayer@rin... X ringlayer@rin... X ringlayer@rin... X ringlayer@rin... X  
ringlayer@ringlayer-Inspiron-3442:~/Desktop/TRAINING/PYTHON/source/2.7$ ./closure_sample.py  
remember me  
ringlayer@ringlayer-Inspiron-3442:~/Desktop/TRAINING/PYTHON/source/2.7$
```

Terlihat kita bisa memanggil inner function dari luar fungsi pembungkusnya dan menampilkan nilai yang berasal dari fungsi outer dengan menggunakan teknik ini.

MetaProgramming

MetaProgramming merupakan teknik pemrograman di mana suatu kode bisa digunakan untuk membaca, memodifikasi / memanipulasi kode. Selain itu, dengan metaprogramming kita bisa menciptakan kode program lain (code generation).

Decorator

Decorator bisa digunakan untuk menambah fungsionalitas dari kode yang sudah ada. Karena decorator digunakan untuk memanipulasi kode yang sudah ada maka decorator merupakan salah satu teknik yang mengimplementasikan metaprogramming. Pada pembahasan kali ini kita akan membahas decorator pada method.

Decorator pada Method

Pada dasarnya decorator pada method dan fungsi memiliki struktur yang serupa dengan closure pada bagian sebelumnya.

Untuk lebih jelasnya, buat contoh file dengan nama decorator_sample.py

```
#!/usr/bin/env python3
"""
sample decorator for python training
at www.jasaplus.com
"""
class Robot():
    def SayHello(self):
        print("[+] hello i am robot")

    def func1(self, func):
        def func2():
            print("[+] decorated")
            self.SayHello()
        return func2

robot = Robot()
res = robot.func1(robot.SayHello)
print(type(res))
res()
```

Penjelasan

```
class Robot():
    def SayHello(self):
        print("[+] hello i am robot")
```

Pada bagian ini kita mempunyai class Robot dengan method SayHello. Method ini nanti akan kita decorated.

```
def func1(self, func):
    def func2():
        print("[+] decorated")
        self.SayHello()
    return func2
```

Method func1 membutuhkan parameter berupa method yang akan didekorasi. Inner class func2 bertujuan untuk mendekorasi method SayHello. Di akhir method nilai kembali berupa objek method func2 dikembalikan.

```
robot = Robot()
res = robot.func1(robot.SayHello)
res()
```

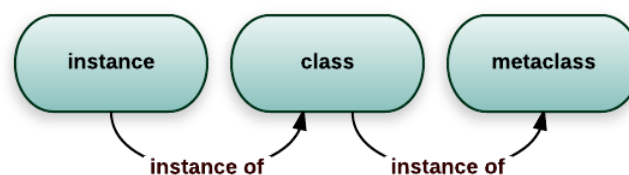
Untuk menguji hasil dekorasi, kita buat instance objek baru dari kelas Robot, kemudian kita panggil method func1 dengan parameter objek fungsi SayHello, hasil dari nilai kembali func1 disimpan pada res, sehingga res merupakan objek instance dari class function.

Jalankan script di atas, hasilnya :

```
ringlayer@ringlayer-Inspiron-3442: ~/Desktop/TRAINING/PYTHON/source/2.7
File Edit Tabs Help
root@ringlay... X ringlayer@rin... X ringlayer@rin... X ringlayer@rin... X ringlayer@rin... X
ringlayer@ringlayer-Inspiron-3442:~/Desktop/TRAINING/PYTHON/source/2.7$ ./decorator_sample.py
<class 'function'>
[+] decorated
[+] hello i am robot
ringlayer@ringlayer-Inspiron-3442:~/Desktop/TRAINING/PYTHON/source/2.7$
```

MetaClass

Metaclass adalah salah teknik di python untuk mengimplementasikan meta programming. Seperti yang kita bahas sebelumnya, di python segala sesuatu (variable, method, function, class) merupakan object. Suatu class di python merupakan object instance dari metaclass. Jadi Metaclass merupakan **class yang instance objectnya merupakan class**.



Metaclass merupakan blueprint dari setiap class yang menjadi instance objectnya. Metaclass ini pada dasarnya merupakan inheritance dari “type”.

Metaclass bergantung pada penggunaan magic method, berikut ini magic method yang biasa digunakan pada suatu metaclass :

1. `__new__`
magic method ini akan dipanggil saat proses konstruksi suatu instance (constructor)
2. `__init__`
magic method ini akan dipanggil setelah object instance terbentuk (initializer)
3. `__call__`
magic method ini akan dipanggil setelah `__init__`

Untuk membuat suatu class yang merupakan instance object dari metaclass, kita bisa memberikan parameter metaclass saat pembuatan class.

Berikut ini pembuatan class baru yang merupakan instance object dari suatu metaclass

Untuk lebih jelasnya, buat file baru dengan nama `metaclass_sample.py` :

```

#!/usr/bin/env python3

"""
metaclass_sample.py
sample metaclass for python3 training
at www.jasaplus.com
"""

#metaclass :
class TheMetaClass:
    #constructor
    def __new__(cls, clsname, superclasses, attributedict):
        try:
            print("[+] __new__")
            print("\tcls : " + str(cls))
            print("\tclsname : " + str(clsname))
            print("\tattribute dict : " + str(attributedict))
            return super().__new__(cls)
        except:
            raise

    #initializer
    def __init__(self, clsname, superclasses, attributedict):
        try:
            print("[+] __init__")
            print(type(self))
        except:
            raise

    #call
    def __call__(self):
        try:
            print("[+] __call__")
            print(type(self))
        except:
            raise

class FooClass(metaclass=TheMetaClass):
    pass

f = FooClass()

```

Jalankan script di atas :

```

ringlayer@ringlayer-Inspiron-3442:~/Desktop/TRAINING/PYTHON/source/2.7$ ./metaclass_sample.py
[+] __new__
      cls : <class ' _main_.TheMetaClass'>
      clsname : FooClass
      attribute dict : {'__qualname__': 'FooClass', '__module__': ' _main_ '}
[+] __init__
<class ' _main_.TheMetaClass'>
[+] __call__
<class ' _main_.TheMetaClass'>
ringlayer@ringlayer-Inspiron-3442:~/Desktop/TRAINING/PYTHON/source/2.7$ █

```

Penjelasan

```
#metaclass :
class TheMetaClass:
    #constructor
    def __new__(cls, clsname, superclasses, attributedict):
        try:
            print("[+] __new__")
            print("\tcls : " + str(cls))
            print("\tclsname : " + str(clsname))
            print("\tattribute dict : " + str(attributedict))
            return super().__new__(cls)
        except:
            raise
```

Pada baris baris di atas kita siapkan suatu metaclass baru dengan nama identifier “TheMetaClass”, selanjutnya kita siapkan method constructor `__new__`

Pada metaclass ini juga terdapat beberapa implementasi magic method :

```
#initializer
def __init__(self, clsname, superclasses, attributedict):
    try:
        print("[+] __init__")
        print(type(self))
    except:
        raise

#call
def __call__(self):
    try:
        print("[+] __call__")
        print(type(self))
    except:
        raise
```

Setelah deklarasi source metaclass dan method method di dalamnya kita coba membuat class baru yang merupakan instance object dari metaclass “TheMetaClass” :

```
class FooClass(metaclass=TheMetaClass):
    pass
```

Setelah class FooClass ready, kita buat instance object baru dari class : “FooClass”

```
f = FooClass()
```

2.8. KUMPULAN TRIK DASAR PYTHON

Berikut ini adalah beberapa kumpulan trik dasar pada python yang akan disajikan pada bagian ini :

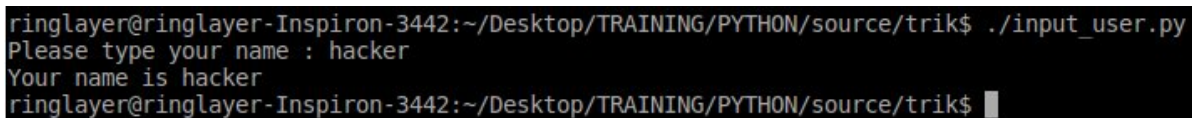
2.8.1. Membaca dan Menampilkan Input dari User

Pada contoh ini, kita akan mencoba membaca input user dan menampilkan inputan tersebut di layar.

Pada python3, untuk menangkap user input kita menggunakan fungsi input().

Buat file python baru dengan nama input_user.py

```
#!/usr/bin/env python3
data = input("Please type your name : ")
print("Your name is {}".format(data))
```



```
ringlayer@ringlayer-Inspiron-3442:~/Desktop/TRAINING/PYTHON/source/trik$ ./input_user.py
Please type your name : hacker
Your name is hacker
ringlayer@ringlayer-Inspiron-3442:~/Desktop/TRAINING/PYTHON/source/trik$
```

Jalankan script di atas, script akan meminta input dari pengguna kemudian menampilkan hasil di layar.

2.8.2. Lambda

Lambda merupakan anonymous function (fungsi tanpa nama) di python. Pada contoh ini, kita akan mencoba penggunaan lambda.

Format penulisan lambda :

lambda arguments : expression

Argumen bisa berjumlah berapapun, expression adalah blok kode singkat dalam 1 baris.

Untuk menguji penggunaan lambda, buat script baru dengan nama lambda_sample.py

```
#!/usr/bin/env python3
"""
lambda samples
for python3 course at jasaplus.com
"""

#sample lambda1
f = lambda x: x+10
print(f(1))
print(f(2))
```

```
print(type(f))

#sample lambda2
f2 = lambda a, b: a+b
print(f2(1, 2))
print(f2(3, 4))
print(type(f2))

#sample lambda2
f3 = lambda a: print(a)
f3("lambda sample")
print(type(f3))
```

Penjelasan

```
f = lambda x: x+10
```

pada baris ini kita mendefinisikan suatu fungsi anonymous dengan parameter variabel x di mana hasil dari x+10 akan disimpan di dalam identifier f bertipe objek fungsi.

```
print(f(1))
```

Pada baris ini, fungsi lambda diwrap dengan fungsi print, di mana hasil dari fungsi f(1) akan ditampilkan di layar dengan fungsi print.

2.8.3. Slicing di Python

Pada contoh ini, kita akan mencoba memahami penggunaan slicing (pengirisan) di python3. Operasi slicing pada string biasanya menggunakan 2 parameter di mana parameter pertama selalu nomor index slice dimulai dan parameter kedua merupakan nomor index kedua slice berakhir (sebelum nomor index kedua ini slice selalu berakhir)

POLA SLICING :

```
string[start:stop:step]
```

di mana :

- start : mulai slice dari indeks ke berapa ?
- stop : slicing hingga nomor index ini -1
- step : angka yang menunjukkan step atau penambahan di antara masing masing indeks slicing, secara default, jika tidak diberikan maka stepnya adalah 1

Misal kita memiliki variable string :

```
data = "ringlayer"
```

Suatu variable string pada di python merupakan kumpulan karakter dengan index.

Jika menggunakan index positif :

0	1	2	3	4	5	6	7	8
r	i	n	g	l	a	y	e	r

Selain menggunakan index positif, kita bisa juga menggunakan index negatif.

Jika menggunakan index negatif :

-9	-8	-7	-6	-5	-4	-3	-2	-1
r	i	n	g	l	a	y	e	r

Misal kita ingin melakukan slicing dengan menggunakan index positif, kita slice dari karakter ke 0 hingga sebelum nomor index ke 3 :

```
print(data[0:3])
```

Hasilnya akan menampilkan string :

```
rin
```

atau bisa juga ditulis seperti ini :

```
print(data[:3])
```

Pada baris di atas dianggap kita akan slicing mulai dari karakter dengan nomor index 0 hingga sebelum nomor index ke 3.

Perhatikan contoh lainnya :

```
print(data[0:-3])
```

Pada contoh di atas kita akan slicing mulai dari karakter ke 0 hingga sebelum nomor index ke -3 pada index negatif.

Hasilnya :

```
ringla
```

Perhatikan contoh lain :

```
print(data[-4:-1])
```

Pada contoh ini, kita slice mulai index ke empat hingga sebelum index ke -1

Hasil :

```
aye
```


[illegible]

- Slice dimulai pada indeks 0
- Slice hingga indeks ke 6 -1
- Slice step adalah 1

Menggunakan step 2 :

```
print(data[0:6:2])
```

hasil :

rnI

Penjelasan :

r	i	n	g	l	a	y	e	r		r	o	b	o	t	i	c
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
r		n		l												

- Slice dimulai pada indeks 0
- Slice hingga indeks ke 6-1
- Slice step adalah 2 indeks

Menggunakan step 3 :

```
print(data[0:6:3])
```

hasil :

rg

Penjelasan :

r	i	n	g	l	a	y	e	r		r	o	b	o	t	i	c
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
r			g													

- Slice dimulai dari index 0
- Slice berhenti pada indeks ke 5 (6-1)
- Step adalah 3 index

Menggunakan step -1 (step terbalik menyebabkan reverse string):

```
print(data[-1:-6:-1])
```

hasil :

citob

Penjelasan :

r	i	n	g	l	a	y	e	r		r	o	b	o	t	i	c
-18	-17	-16	-15	-14	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
												b	o	t	i	c

- Slice dimulai pada index -1
- Slice hingga -6 -1 = -5
- Step adalah -1 (berjalan mundur sebanyak tiap 1 index)

2.8.4. Membalikkan String

Reverse string merupakan salah satu trik yang mungkin berguna, digunakan untuk membalikkan suatu string, contoh string : good , setelah direverse menjadi doog.

Berikut ini contoh :

```
#!/usr/bin/env python3
a = "ringlayer"
print(a[::-1])
```

Hasil :
reyalgnir

Penjelasan

Step adalah -1, di mana start dan stop tidak didefinisikan, secara otomatis data akan dislice terbalik (reverse string).

r	i	n	g	l	a	y	e	r
-9	-8	-7	-6	-5	-4	-3	-2	-1

Berikut ini source code lengkap contoh kedua (slicing2.py) :

```
#!/usr/bin/env python3
data = "ringlayer robotic"
```

```
#ringla
print(data[0:6])
```

```
#ringla
print(data[0:6:1])
```

```
#rnl
print(data[0:6:2])
```

```
#rg
print(data[0:6:3])
```

```
#reverse step result : citob
print(data[-1:-6:-1])
```

2.8.5. Argumen Perintah di Python

Pada contoh ini, kita akan mencoba melakukan pembacaan argumen perintah yang diinput saat menjalankan script python. Python menyediakan pembacaan argumen perintah seperti pada getopt dengan c, di mana penanganan argumen perintah diimplementasi pada modul sys.

Untuk memulai, lakukan import modul sys : `import sys`. Setelah modul sys diimport maka argumen perintah yang diberikan pada script akan disimpan pada list `sys.argv`.

Misal anda memiliki script nama `nama_script.py`, kemudian menjalankan script dengan argumen seperti ini :

```
python3 nama_script.py arg1 arg2 arg3
```

maka list `sys.argv` akan berisi seperti ini :

<code>sys.argv[0]</code>	<code>sys.argv[1]</code>	<code>sys.argv[2]</code>	<code>sys.argv[3]</code>
<code>nama_script.py</code>	<code>arg1</code>	<code>arg2</code>	<code>arg3</code>

Berikut ini contoh script dengan pembacaan argumen perintah, buat script baru dengan nama `argument_sample.py` :

```
#!/usr/bin/env python3
"""
sample argument processing demo
for python3 course at jasaplus.com
"""
import sys
print("\n\tType of sys.argv : ", type(sys.argv))
print("\n\tLength of sys.argv : ", len(sys.argv))
print("\n\tPrinting each argument :")
i = 0
for x in sys.argv:
    print("\n\tsys.argv[" + str(i).strip() + "] is : " + sys.argv[i])
    i+=1
```

Penjelasan

```
print("\n\tLength of sys.argv : ", len(sys.argv))
```

Baris ini digunakan untuk menampilkan jumlah argumen perintah, termasuk nama script

```
i = 0
for x in sys.argv:
    print("\n\tsys.argv[" + str(i).strip() + "] is : " + sys.argv[i])
    i+=1
```

Rutin di atas akan menampilkan seluruh isi list `sys.argv`