MODUL 3

Python Enthusiast

```
bootcamp = [“ParselMouth”, “PythonEnthusiast”]
>> for i in bootcamp:
>>     if i:
>>         print(“Hello {}”.format(i))
...
```

>> print(“let’s learn Python Idiom”)

pwcahyo@gmail.com | abie.mb5@gmail.com

Python Idiom

Seorang Python Enthusiast harus *"Smart"*, definisi *smart* kali ini bukan hanya tertuju pada kasus/masalah sudah bisa terselesaikan, akan tetapi lebih ke cara bagaimana kita bisa menyelesaikan kasus tersebut, kita bergerak dengan cara yang rapi atau hanya asal asal - asalan. Seperti halnya orang membangun rumah, apakah rumah yang dibangun cukup hanya digunakan untuk berteduh dan tidur ?. Tidakkah dilihat dari segi estetika, bentuk, warna, kesopanan sehingga menyatu menjadi wujud yang indah ?.

Python idiom merupakan salah satu kebanggaan yang dimiliki oleh Python Enthusiast, didalam Python idiom tersebut programmer dilatih untuk bisa menerapkan style kode yang rapi dan indah, sehingga kode program yang ditulis menjadi lebih enak dibaca dan tanpa memerlukan banyak resource. Berikut ini dijelaskan beberapa Style dari Python Idiom yang dapat kita terapkan.

1. Make a script both importable and executable using **if __name__ == "__main__":** (Membuat kode importable dan executable)

   Bad

```
def main():
    print("Doing stuff in module", __name__)

print('Executed from the command line')
main()
```

   Good

```
def main():
    print("Doing stuff in module", __name__)

if __name__ == "__main__":
    print("Doing stuff in module", __name__)
    main()
```

   Note: Doing stuff in module mymodule

2. Chained Comparison Operators (Operator perbandingan secara berantai)

   Bad

```
if x <= y and y <=z:
        #do something
```

   Good

```
if x <= y <=z:
        #do something
```

3. Identation and Semicolon (Penggunakan indentasi dan titik koma)

Bad

```
name = “PWCahyo”; address = “Sleman”
if name: print(name)
print(address)
```

Good

```
name = "PWCahyo"
address = "Sleman"
if name:
        print(name)
print(address)
```

4. Use the Falsy & Truthy Concepts (Penggunakan Konsep True False)

Bad

```
name = "PWCahyo"
pets = ["Python", "Anaconda", "Rat"]
owners = {"PWCahyo": "Python", "Habibi": "Rat"}
if name != “” and len(pets) > 0 and owners != {}:
    print(“Kita Punya Hewan Peliharaan”)
```

Good

```
name = "PWCahyo"
pets = ["Python", "Anaconda", "Rat"]
owners = {"PWCahyo": "Python", "Habibi": "Rat"}
if name and pets and owners:
  print("Kita Punya Hewan Peliharaan")
```

Note:

- Checking for truth doesn't tie the conditional expression to the type of object being checked.
- Checking for truth clearly shows the code's intention rather than drawing attention to a specific outcome.

5. Using *in* possible contain or iteration (checking apakah ada karakter huruf atau data ada pada list)

Bad

```
name = "PWCahyo"
  if name.find("C") != -1:
    print("Huruf C ada")
```

Good

```
name = "PWCahyo"
  if “C” in name:
    print("Huruf C ada")
```

Note:

- Using in to check if an item is in a sequence is clear and concise.
- Can be used on lists, dicts (keys), sets, strings, and your own classes by implementing the __contains__ special method.

(melakukan iterasi data pada list)

Bad

```
pets = ["Python", "Anaconda", "Rat"]
i=0
while i < len(pets):
    print('Seekor', pets[i], 'Mengejutkan')
    i += 1
```

Good

```
pets = ["Python", "Anaconda", "Rat"]
for pet in pets:
    print('Seekor', pets[i], 'Mengejutkan')
```

Note:

- Using in to for iteration over a sequence is clear and concise.
- Can be used on lists, dicts (keys), sets, strings, and your own classes by implementing the __iter__ special method.

6. Swap values without temp variable (pertukaran nilai tanpa variable temporary)

Bad

```
x, y = 2, 9
print(x, y)
# 2, 9

temp = x
x=y
y = temp
print(x, y)
# 9, 2
```

Good

```
x, y = 2, 9
print(x, y)
# 2, 9

x, y = y, x
print(x, y)
# 9, 2
```

Note : Avoids polluting namespace with temp variable used only once.

7. Build strings using sequence

Bad

```
chars = ["C", "a", "h", "y", "o"]
name = ""
for char in chars:
    name += char
print(name)          # Cahyo
```

Good

```
chars = ["C", "a", "h", "y", "o"]
name = “”.join(chars)
print(name)          # Cahyo
```

Note:

- The join method called on a string and passed a list of strings takes linear time based on length of list.
- Repeatedly appending to a string using '+' takes quadratic time!

8. EAFP is preferable to LBYL

“It's **E**asier to **A**sk for **F**orgiveness than **P**ermission.”

“**L**ook **B**efore **Y**ou **L**eap”

Kebiasaan manusia:

“tidak mau berhati – hati terlebih dahulu,

kalau sudah melakukan kesalahan baru meminta maaf”. *_dasar manusia*

Bad

```
you = {"jump": "whoaaa"}
if "jump" in you and \
        isinstance(you["jump"], str) and \
        you["jump"].isdigit():
        value = int(you["jump"])
else:
        value = None
```

Good

```
you = {"jump": "whoaaa"}
try:
        value = int(you['jump'])
except (KeyError, TypeError, ValueError):
        value = None
```

Note:

- Throwing exceptions is not “expensive” in Python unlike e.g. Java.
- Rely on duck typing rather than checking for a specific type.

9. Using *enumerate*

   (Penggunakan enumerate pada perulangan, untuk mendapatkan index data)

   Bad

```
names = ["Cahyo", "Habibi", "Dana", "Aris"]
for name in names:
    print(i, name)
    count += 1
```

   Good

```
names = ["Cahyo", "Habibi", "Dana", "Aris"]
for i, name in enumerate(names):
    print(i, name)
```

10. Build lists using list comprehensions

    (Menciptakan list dengan list comprehensions)

    Bad

```
data = [7, 20, 3, 15, 11]
result = []
for i in data:
    if i > 10:
        result.append(i * 3)
print(result)   # [60, 45, 33]
```

    Good

```
data = [7, 20, 3, 15, 11]
result = [i * 3 for i in data if i > 10]
print(result)   # [60, 45, 33]
```

Note :

- Very concise syntax.
- Be careful it doesn't get out of hand (in which case the first form can be clearer).

11. Dan sisanya adalah:
    1) while True:
        break # This will spark discussion!!!
    2) Generators and generator expressions.
    3) Avoid from module import *
       Prefer: import numpy as np; import pandas as pd
    4) Use _ for “throwaway” variables e.g.:
       for k, _ in [('a', 1), ('b', 2), ('c', 3)] dict.get() and
    5) dict.setdefault()
    6) collections.defaultdict
    7) Sort lists using l.sort(key=key_func)