**Rangkuman Style Guide pada Python**

Kita sudah berada di penghujung materi style guide pada Python. Sampai sejauh ini, Anda diharapkan paham untuk memahami cara membangun kode Python yang lebih baik dan benar sesuai panduan yang telah diberikan oleh Python melalui PEP8. Mari kita rangkum secara saksama.

**Pengecekan Style Guide PEP8**

Saat membangun program pada Python, seringkali kode yang dibuat 'cukup berantakan' sehingga kita perlu mengecek bahwa kode tersebut sudah rapi dan benar dengan mengacu pada panduan yang telah diberikan oleh Python, yaitu PEP 8.

PEP atau *Python Enhancement Proposals* adalah panduan yang telah menjadi acuan untuk perkembangan Python. Salah satu panduan tersebut membahas mengenai arahan gaya penulisan (style guide) yang baik dan benar ketika Anda ingin membangun kode menggunakan Python. Panduan tersebut adalah PEP8 yang berjudul "[Style Guide for Python Code](https://peps.python.org/pep-0008/)".

Tujuan dari panduan ini agar kode Anda lebih mudah dibaca dan dipahami oleh programmer lain serta menghindari kemungkinan kesalahan yang akan muncul.

**Lint**

Lint atau linting adalah proses pengecekan kode atas kemungkinan terjadi kesalahan (error), termasuk dalam proses ini adalah mengecek kesesuaian terhadap arahan gaya penulisan kode (style guide) PEP8. Aplikasi yang digunakan untuk proses ini disebut linter. Integrasi linter dengan editor kode Anda akan membuat efisien dalam menulis kode Python. Pertimbangan ini karena keluaran atau output dari aplikasi linter hanya berupa teks singkat berupa keterangan dan kode Error atau Warning atau Kesalahan Konvensi Penamaan (Naming Conventions).

Dengan lint atau linting akan meminimalkan kode Anda mengalami error, salah satunya karena kesalahan indentasi di Python. Sebelum kode Anda diproses oleh interpreter Python dengan IndentationError, lint akan memberitahukannya lebih dahulu ke Anda. Berikut adalah tiga jenis aplikasi linter.

1. **Pycodestyle** (sebelumnya bernama pep8)  
   [Pycodestyle](https://github.com/PyCQA/pycodestyle) adalah aplikasi open source (berlisensi MIT/Expat) untuk membantu mengecek kode terkait gaya penulisan kode dengan konvensi PEP8.
2. **Pylint**  
   [Pylint](https://github.com/PyCQA/pylint) adalah aplikasi open source (berlisensi GPL v2) untuk melakukan analisis kode Python, mengecek untuk kesalahan (error) pemrograman, memaksakan standar penulisan kode dengan mengecek penulisan kode yang tidak baik, serta memberikan saran untuk refactoring sederhana.
3. **Flake8**  
   [Flake8](https://github.com/PyCQA/flake8) adalah aplikasi open source (berlisensi MIT) yang membungkus sejumlah kemampuan aplikasi lain, seperti pycodestyle, pyflakes, dan sejumlah (skrip/fitur) lainnya.

**Memformat Kode**

Jika proses lint atau linting hanya melakukan pengecekan, kali ini adalah arahan gaya penulisan kode agar bisa sesuai dengan PEP8. Kita akan kembali menggunakan beberapa aplikasi yang nantinya akan diinstal.

Proses memformat kode akan sama dengan cara melakukan proses linting, yaitu kita akan mengeksekusi script. Perbedaannya adalah output yang dihasilkan. Jika proses linting menghasilkan pesan dengan menunjukkan baris dan kode yang mengalami kesalahan, proses memformat kode akan memberikan pesan berupa kode yang telah diperbaiki. Ini artinya Anda tidak perlu mengubah kode secara manual. Berikut adalah tiga jenis aplikasi untuk memformat kode.

1. **black**  
   [black](https://github.com/psf/black) adalah proyek open source yang dikembangkan di repository Python Software Foundation (PSF) dengan lisensi MIT. Untuk mendapatkan gambaran, versi online (tidak resmi) ada di [https://black.now.sh](https://black.now.sh/).
2. **YAPF (Yet Another Python Formatter)**  
   [YAPF](https://github.com/google/yapf) adalah proyek open source yang dikembangkan di repository Google dengan lisensi Apache.
3. **autopep8**  
   [autopep8](https://github.com/hhatto/autopep8) adalah proyek open source (berlisensi MIT) yang termasuk paling awal untuk memformat kode dengan bantuan lint pycodestyle.

**Style Guide Statement Gabungan**

Setelah mengetahui aplikasi untuk pengecekan dan memformat kode, kali ini kita akan belajar cara membuat kode yang baik dan benar.

**Statement Gabungan**

Saat Anda membuat program dengan banyak statement, usahakan untuk tidak menggabungkan >1 statement pada baris yang sama.

**Disarankan** seperti ini.

1. if foo == 'blah':
2. do\_blah\_thing()
3. do\_one()
4. do\_two()
5. do\_three()

**Tidak** disarankan seperti ini.

1. if foo == 'blah': do\_blah\_thing()
2. do\_one(); do\_two(); do\_three()

Anda diperbolehkan untuk membuat sebuah konten/isi dari if/for/while yang cukup pendek untuk diletakkan dalam satu baris (program tetap berjalan). Namun, pastikan tidak melakukannya jika if/for/while Anda bertingkat atau bersifat multi clause, misalnya if-else, try-finally, dan sebagainya.

**Tidak** disarankan seperti ini.

1. if foo == 'blah': do\_blah\_thing()
2. for x in lst: total += x
3. while t < 10: t = delay()

**Sangat tidak** disarankan seperti ini.

1. if foo == 'blah': do\_blah\_thing()
2. else: do\_non\_blah\_thing()
3. try: something()
4. finally: cleanup()
5. do\_one(); do\_two(); do\_three(long, argument,
6. list, like, this)
7. if foo == 'blah': one(); two(); three()

**Penggunaan Trailing Commas**

Koma di bagian akhir (trailing commas) umumnya bersifat opsional, satu statement ketika ia bersifat wajib adalah saat kita membuat variabel menggunakan tipe tuple dengan satu elemen. Hal ini umumnya diperjelas dengan kurung untuk menghindari penghapusan atau pembersihan.

**Disarankan** seperti ini.

1. FILES = ('setup.cfg',)

**Tidak** disarankan seperti ini.

1. FILES = 'setup.cfg',

Tidak umum jika Anda meletakkan trailing comma pada baris tempat Anda menutup kurung/kurawal/siku seperti di bawah ini, kecuali dalam tuple dengan satu elemen seperti yang dijelaskan di atas.

**Disarankan** seperti ini.

1. FILES = [
2. 'setup.cfg',
3. 'tox.ini',
4. ]
5. initialize(FILES,
6. error=True,
7. )

**Tidak** disarankan seperti ini.

1. FILES = ['setup.cfg', 'tox.ini',]
2. initialize(FILES, error=True,)

**Anotasi Fungsi**

Anotasi fungsi adalah fitur yang memungkinkan kita untuk menambahkan informasi tambahan tentang parameter dan return value dari sebuah fungsi. Jika sebelumnya kita belajar menambahkan informasi terkait fungsi dengan menambahkan docstring, anotasi fungsi lebih spesifik untuk menjelaskan parameter dan return value.

Penggunaan anotasi fungsi sebaiknya menggunakan aturan baku untuk titik dua (:) dan menggunakan spasi untuk penggunaan arah panah atau arrow (->). Hal ini disebut sebagai type hints yang merujuk pada [PEP 484](https://www.python.org/dev/peps/pep-0484).

1. Yes:
2. def munge(input: str):  # Menambahkan informasi parameter bertipe string
3. pass
4. def munge() -> str:   # Menambahkan informasi return value bertipe string
5. pass
7. No:
8. def munge(input:str):  # Menambahkan informasi parameter bertipe string
9. pass
10. def munge()->str:   # Menambahkan informasi return value bertipe string
11. pass

Jika kita membuat fungsi yang menggabungkan anotasi dengan nilai parameter, sebaiknya tetap menggunakan spasi sebelum dan sesudah tanda sama dengan (=). Namun, ketika membuat fungsi biasa tanpa adanya anotasi, sebaiknya tidak menggunakan spasi sebelum dan sesudah tanda sama dengan (=).

1. Yes:
2. def LuasPersegiPanjang(panjang:int = 2, lebar=None):
3. pass
5. No:
6. def LuasPersegiPanjang(panjang: int=2, lebar = None):
7. pass

**Style Guide Prinsip Penamaan pada Python**

Penamaan pada Python mencakup banyak hal, seperti penamaan fungsi, kelas, dan sebagainya. Berikut adalah beberapa rekomendasi untuk penamaan yang meliputi overriding, penamaan deskriptif, fungsi, method, dan sebagainya. Namun, Anda juga dapat memilih mempertahankan styling yang sudah digunakan sebelumnya untuk menjaga konsistensi internal tim atau perusahaan. Ini karena konsistensi internal lebih diutamakan.

**Prinsip Overriding**

Nama yang dilihat oleh user publik sebaiknya merefleksikan penggunaan/fungsinya dan bukan implementasinya.

**Penamaan Deskriptif**

Penamaan deskriptif adalah cara untuk memberikan nama yang informatif, jelas, dan sesuai dengan tujuan dari elemen kode. Penamaan deskriptif ini meliputi variabel, fungsi, kelas, hingga konstanta.

Ada berbagai cara penamaan yang umum digunakan dalam Python. Pemilihan cara penamaan ini penting untuk menjaga konsistensi dan kejelasan kode. Penamaan ini juga merujuk pada PEP8 mengenai [Naming Conventions](https://peps.python.org/pep-0008/#naming-conventions) dan [Naming Styles](https://peps.python.org/pep-0008/#descriptive-naming-styles).

Berikut adalah beberapa cara penamaan yang umum:

1. Satu karakter huruf kecil: b
2. Satu karakter huruf besar: B
3. Huruf kecil: hurufkecil
4. Huruf kecil dengan pemisah kata garis bawah: huruf\_kecil\_dengan\_pemisah\_kata\_garis\_bawah
5. HURUF BESAR: HURUFBESAR
6. HURUF BESAR dengan pemisah garis bawah: HURUF\_BESAR\_DENGAN\_PEMISAH\_GARIS\_BAWAH
7. Huruf Besar di Awal Kata (CapWords, CamelCase): HurufBesarDiAwalKata (pastikan semua singkatan/akronim dituliskan dengan huruf besar, misalnya HTTPServerError, bukan HttpServerError)
8. Huruf Campuran: hurufCampuran (mirip dengan CapWords, hanya berbeda di karakter paling awal)
9. Huruf Besar di Awal Kata dengan Garis Bawah: Huruf\_Besar\_Di\_Awal\_Kata\_Dengan\_Garis\_Bawah

Python tidak menyarankan atau lebih tepatnya tidak dibutuhkan jika Anda membuat sebuah fungsi yang diawali huruf atau frasa, seperti 'f' jika fungsinya 'f\_mean()',  'r' jika fungsinya 'r\_name()', dan sebagainya.

Selain penggunaan huruf atau frasa yang tidak direkomendasikan, berikut adalah beberapa bentuk penamaan khusus yang umum ditemukan dalam penamaan fungsi. Ini juga bisa Anda terapkan pada penamaan variabel dan kelas.

1. \_diawali\_sebuah\_garis\_bawah: penamaan ini dapat digunakan untuk penggunaan internal lemah yang merujuk pada penggunaannya dengan lingkup tertentu.
2. diakhiri\_sebuah\_garis bawah\_: penamaan ini digunakan untuk mengatasi redundan dengan *keyword*/*reserved words* di Python.
3. \_\_diawali\_dua\_garis bawah: menegaskan bahwa sebuah objek adalah bagian dari kelas tertentu.
4. \_\_diawali\_dan\_diakhiri\_dua\_garis bawah\_\_: Objek atau atribut tertentu yang diciptakan Python untuk digunakan dalam program. Contohnya adalah  \_\_init\_\_, \_\_import\_\_ or \_\_file\_\_.

**Hal-Hal yang Harus Diperhatikan dalam Penamaan**

Berikut adalah hal-hal yang harus diperhatikan dalam penamaan fungsi, method, hingga penamaan kelas.

1. **Nama yang Dihindari**  
   Hindari karakter l (huruf L kecil), O (huruf o besar) atau I (huruf i besar) sebagai nama variabel satu karakter karena mereka sulit dibedakan dengan angka satu dan nol.
2. **ASCII Compatibility**  
   Merujuk pada [PEP 3131](https://www.python.org/dev/peps/pep-3131), suatu identifiers yang digunakan dalam Python Standard Library harus kompatibel dengan kode ASCII.
3. **Nama Paket dan Nama Modul**  
   Penamaan modul sebaiknya pendek atau singkat, menggunakan huruf kecil, dan opsional garis bawah (\_) untuk meningkatkan keterbacaan. Nama paket juga sebaiknya singkat, menggunakan huruf kecil, dan hindari garis bawah(\_).
4. **Nama Kelas**  
   Saat menamai kelas, gunakan CamelCase atau CapWords. Pastikan semua akronim (misal HTTP) ditulis keseluruhan dengan huruf besar.
5. **Penulisan Tipe Variabel**  
   Untuk penamaan variabel, umumnya menggunakan CamelCase atau CapWords.
6. **Nama Exception**  
   Untuk pengecualian (exception), Anda juga menerapkan konvensi penamaan kelas pada exception karena ia seharusnya bertipe kelas. Bedanya, tambahkan "Error" atau nama deskriptif lain pada nama exception Anda.
7. **Nama Variabel Global**  
   Dalam variabel global, penamaannya bisa mengikuti fungsi/modul yang bersifat publik. Anda bisa menggunakan garis bawah untuk menghindari variabel tersebut diimpor jika ia termasuk modul non-publik.
8. **Nama Fungsi, Parameter, dan Variabel**  
   Nama fungsi, parameter, dan variabel sebaiknya menggunakan huruf kecil dengan pemisahan menggunakan garis bawah untuk meningkatkan keterbacaan. mixedCase dapat digunakan jika ada dependensi dengan pustaka menggunakan style tertentu.
9. **Argumen Fungsi dan Method**  
   Dalam pembuatan fungsi dan method pada suatu kelas, ada beberapa hal yang perlu dipertimbangkan..
   1. Gunakan self sebagai argumen pertama jika Anda membuat instance method.
   2. Gunakan cls sebagai argumen pertama ketika Anda membuat class method.
10. **Nama Method dan Variabel Instance**  
    Saat membuat method dan variabel dalam suatu kelas, gunakan standar penamaan fungsi, yaitu gunakan huruf kecil dengan pemisah kata garis bawah untuk meningkatkan keterbacaan. Tambahkan garis bawah sebagai awalan untuk method non-publik dan variabel internal pada fungsi.
11. **Konstanta**  
    Dalam memberikan nama variabel bertipe konstanta, umumnya didefinisikan pada bagian atas modul dengan huruf besar, misalnya 'PI = 3,14'  atau  'TOTAL = 4.14213'.
12. **Selalu Persiapkan untuk Inheritance**  
    Saat membangun metode dan variabel dalam sebuah kelas, sebaiknya Anda dapat langsung mengetahui atribut pada metode dan variabel tersebut, publik atau non-publik. Jika Anda ragu, jadikan atributnya non-publik. Sebab, lebih mudah menjadikan sebuah variabel/method bersifat non-publik menjadi publik, dibandingkan sebaliknya.  
      
    Variabel atau method yang bersifat non-publik adalah suatu variabel atau method yang hanya digunakan untuk lingkup tertentu dan tidak diakses secara langsung di luar.  
      
    Method/Variabel publik dipersiapkan untuk pihak eksternal menggunakan kelas Anda. Anda juga otomatis berkomitmen untuk menghindari adanya *incompatible backward changes*atau suatu kode yang tidak dapat berjalan kembali setelah adanya perubahan.  
      
    Sebaliknya, method/variabel dengan atribut non-publik hanya digunakan oleh Anda sebagai developer. Itu juga tidak memberikan garansi kepada siapa pun bahwa Anda takkan mengubah atau menghapusnya. Di sini kita tidak menggunakan atribut private karena dalam Python tidak ada atribut yang benar-benar private.  
      
    Saat mendeklarasikan variabel/method tersebut, ikuti panduan Pythonic berikut.
    1. Atribut publik tidak menggunakan awalan garis bawah.
    2. Jika nama sebuah method/variabel publik sama dengan *reserved keyword*, tambahkan akhiran garis bawah. Hindari menyingkat atau mengurangi huruf.
    3. Pada data publik bersifat simpel, hindari nama yang terlalu panjang. Cukup dengan nama atribut sependek mungkin. Ingatlah bahwa pada masa depan Anda akan mungkin mengembangkan skema atau data ini sehingga nama sependek apa pun mungkin akan menguntungkan Anda.
    4. Jika Anda berniat untuk mewariskan atau membuat subclass dari kelas dan menginginkan sebuah variabel hanya digunakan di kelas utama saja, tambahkan awalan dua garis bawah. Ini akan memudahkan Anda karena Python mengenalinya sebagai konvensi kelas, untuk menghindari kemungkinan kesamaan nama atau implementasi.