**Pengecekan Style Guide PEP8**

Sampai dengan saat ini, Anda tentu sudah menuliskan kode pemrograman Python cukup banyak, termasuk di antaranya kode yang Anda tulis mengalami kesalahan sintaksis atau indentasi. Lalu, bagaimana agar ke depannya bisa mencegah hal tersebut terjadi?

Semua itu bergantung pada kode editor yang Anda gunakan untuk menulis kode Python, terkadang ada beberapa kode editor yang sudah dilengkapi dengan fitur pengecekan kemungkinan kesalahan dan memformat kode sesuai arahan gaya penulisan (style guide) PEP8, seperti PyCharm. Ada juga kode editor, seperti Visual Studio Code yang menyediakan plugin tambahan untuk membantu pengecekan kemungkinan kesalahan dan memformat kode.

PEP atau *Python Enhancement Proposals* merupakan 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.

Pada materi kali ini, kita akan mempelajari beberapa aplikasi yang dapat Anda gunakan dengan memanggil perintah atau sebaiknya diintegrasikan ke editor kode yang Anda pakai. Tujuannya untuk membantu Anda mengecek kemungkinan kesalahan dan kesesuaian dengan PEP8.

**Lint**

Lint 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).

Lint atau linting akan meminimalkan kode Anda mengalami error, salah satu contohnya karena kesalahan indentasi di Python. Sebelum kode Anda diproses oleh interpreter Python dengan IndentationError, lint akan memberitahukannya lebih dahulu ke Anda.

Berikut akan dibahas tiga jenis aplikasi linter, silakan dicermati dahulu. Tidak harus semuanya diinstal/dicoba, hanya paket yang menurut Anda sesuai kebutuhan saja yang digunakan. Untuk menginstalnya, silakan buka terminal Anda dan jalankan kode di bawah ini sesuai yang Anda pilih.

**Catatan:** Output ketiga aplikasi ini kemungkinan mirip, tetapi pada kondisi tertentu akan ada output atau fitur yang mungkin sesuai dengan kebutuhan Anda menulis kode.

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 (skrip/fitur) lainnya.

Selanjutnya, mari kita masuk ke pembahasan cara kerja ketiganya. Pastikan Anda sudah menginstal aplikasi yang disebutkan sebelumnya.

1. Masuk ke kode editor Anda, misalnya Visual Studio Code.
2. Buat sebuah file bernama kalkulator.py dan masukkan kode berikut.

|  |
| --- |
| class Kalkulator:  """kalkulator tambah kurang"""  def \_\_init\_\_(self, \_i):  self.i = \_i  def tambah(self, \_i): return self.i + \_i  def kurang(self, \_i):  return self.i - \_i |

1. Pada kode di atas, kita membuat kelas bernama Kalkulator. Kelas ini memiliki dua metode, yaitu tambah dan kurang. Atribut yang dimiliki kelas ini hanyalah variabel "i".  
     
   Berdasarkan PEP8, kode tersebut masih perlu diperbaiki dan ada blok kode yang akan menghasilkan error. Kita akan mengetahuinya nanti.
2. Mari kita jalankan file atau script tersebut dengan aplikasi yang telah diinstal. Buka kembali terminal Anda, pastikan membuka direktori tempat file atau script Anda berada.

**Pycodestyle**  
Untuk menguji menggunakan pycodestyle, jalankan kode berikut.

|  |
| --- |
| pycodestyle kalkulator.py |



**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, silakan dicermati dahulu. Tidak harus semuanya diinstal, hanya paket yang menurut Anda sesuai kebutuhan saja yang digunakan.

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/" \t "_blank).
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.

Selanjutnya, mari kita masuk ke pembahasan cara kerja ketiganya. Kita akan menggunakan kode yang sama seperti sebelumnya, yaitu kalkulator.py. Pastikan Anda sudah menginstal aplikasi yang disebutkan sebelumnya.

1. Buka script kalkulator.py dan salin kode berikut.

|  |
| --- |
| class Kalkulator:  """kalkulator tambah kurang"""  def \_\_init\_\_(self, \_i):  self.i = \_i  def tambah(self, \_i): return self.i + \_i  def kurang(self, \_i):  return self.i - \_i |

1. Contoh di atas merupakan kode kalkulator yang sama seperti materi sebelumnya, saat kita mendefinisikan kelas kalkulator dengan dua metode, yaitu tambah dan kurang serta satu atribut objek.
2. Mari jalankan file atau script tersebut dengan aplikasi yang telah kita instal. Buka kembali terminal Anda, pastikan membuka direktori tempat file atau script Anda berada.
   1. **black**
   2. **yapf**
   3. **autopep8**

Setelah mempelajari kedua mekanisme: pengecekan gaya penulisan (style guide) dan proses memformat kode, Anda tinggal fokus dengan penulisan indentasi dalam kode Python. Untuk format sisanya, aplikasi-aplikasi yang telah kita pelajari di atas dapat membantu.

Jika Anda menulis dengan editor kode yang sangat sederhana, anggap saja seperti notepad di Windows atau pico/nano di Linux, dalam menuliskan kode Python cukup perhatikan indentasi untuk setiap baris pernyataan (statement). Setelah selesai menuliskan kodenya, simpan kodenya sebagai file .py, lalu eksekusi perintah linter atau langsung eksekusi perintah aplikasi untuk memformat kode. Hasilnya, kode Anda sudah dirapikan sesuai arahan gaya penulisan PEP8 juga dilakukan pengecekan terhadap kemungkinan kesalahan (jika Anda lakukan linting).

Untuk Anda yang mengikuti instalasi editor kode PyCharm dalam materi awal, secara bawaan sudah menggunakan fitur inspeksi dengan kemampuan yang kurang lebih sama. Namun jika Anda mau, bisa juga menambahkan aplikasi lint yang sudah dijelaskan sebagai tambahan dari bawaan. Demikian juga untuk fitur format ulang kode juga tersedia secara bawaan di aplikasi PyCharm.

**Style Guide Statement Gabungan**

Setelah mengetahui aplikasi untuk pengecekan dan memformat kode, kali ini kita akan belajar cara membuat kode yang baik dan benar. Perhatikan bahwa materi ini akan menunjukkan sintaks yang disarankan dan tidak disarankan.

**Statement Gabungan**

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

|  |
| --- |
| if foo == 'blah':  do\_blah\_thing()  do\_one()  do\_two()  do\_three() |

**Tidak** disarankan seperti ini.

|  |
| --- |
| if foo == 'blah': do\_blah\_thing()  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.

|  |
| --- |
| if foo == 'blah': do\_blah\_thing()  for x in lst: total += x  while t < 10: t = delay() |

**Sangat tidak** disarankan seperti ini.

|  |
| --- |
| if foo == 'blah': do\_blah\_thing()  else: do\_non\_blah\_thing()  try: something()  finally: cleanup()  do\_one(); do\_two(); do\_three(long, argument,  list, like, this)  if foo == 'blah': one(); two(); three() |

**Penggunaan Trailing Commas**

Koma di bagian akhir (*trailing commas*) umumnya bersifat opsional, satu statement yang 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.

|  |
| --- |
| FILES = ('setup.cfg',) |

**Tidak**disarankan seperti ini.

|  |
| --- |
| FILES = 'setup.cfg', |

Saat trailing comma bersifat redundan, Anda akan merasakan kemudahannya saat menggunakan VCS (Version Control System), atau pada kode yang mungkin ditambahkan dalam beberapa waktu ke depan. Pola yang disarankan adalah meletakkan nilai atau string pada sebuah baris baru, mengikuti indentasi, menambahkan trailing comma, dan menutup kurung/kurawal/siku pada baris selanjutnya.

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

Disarankan seperti ini.

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

**Tidak** disarankan seperti ini.

|  |
| --- |
| FILES = ['setup.cfg', 'tox.ini',]  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).

|  |
| --- |
| # Perhatikan penggunaan spasi dari kedua kode berikut.    Yes:  def munge(input: str): # Menambahkan informasi parameter bertipe string  pass  def munge() -> str: # Menambahkan informasi return value bertipe string  pass    No:  def munge(input:str): # Menambahkan informasi parameter bertipe string  pass  def munge()->str: # Menambahkan informasi return value bertipe string  pass |

Pada contoh di atas, kita memberikan informasi bahwa parameter dan return value harus berupa tipe data string. Kita bisa menentukannya dengan tipe lain, seperti 'int' untuk integer dan 'float' untuk tipe data float.

Selanjutnya, saat membuat fungsi dan Anda menggabungkan anotasi dengan nilai parameter, sebaiknya tetap menggunakan spasi baik sebelum dan sesudah tanda sama dengan (=).

|  |
| --- |
| def LuasPersegiPanjang(panjang: int = 2, lebar: int = None):  pass |

Pada contoh di atas, kita membuat fungsi bernama "LuasPersegiPanjang" untuk mencari luas persegi panjang dengan parameter panjang dan lebar. Sintaks berikut menjelaskan bahwa parameter panjang dan lebar harus bertipe data integer.

|  |
| --- |
| panjang: int |

Sementara itu, ketika menambahkan variabel setelah sama dengan (=) akan memberikan nilai default. Contohnya sintaks berikut akan memberikan nilai default 2 untuk parameter panjang.

|  |
| --- |
| panjang: int = 2 |

|  |
| --- |
| def LuasPersegiPanjang(panjang: int = 2, lebar: int = None):  luas = panjang\*lebar  return luas  luas\_satu = LuasPersegiPanjang(lebar=2)  print(luas\_satu)  Output:  4 |

Pada contoh di atas, kita membuat fungsi untuk mencari luas persegi panjang dengan parameter panjang dan lebar. Perlu diingat bahwa pada fungsi tersebut kita memberikan nilai default 2 untuk parameter panjang. Hal ini mengakibatkan bahwa ketika memanggil fungsi LuasPersegiPanjang dengan hanya memasukkan argumen lebar, program akan tetap berjalan dengan baik.

Namun, perlu diingat bahwa karena type hints bersifat optional dan memberikan petunjuk, jika pada fungsi LuasPersegiPanjang kita memberikan tipe data float, program akan tetap berjalan sebagaimana mestinya.

Sekarang, kita masuk ke skenario baru. Jika pada saat membuat fungsi tanpa adanya anotasi bahwa parameternya menandakan keyword argumen atau nilai default, hindari penggunaan spasi di sekitar tanda sama dengan (=).

|  |
| --- |
| Yes:  def LuasPersegiPanjang(panjang=2, lebar=None):  luas = panjang\*lebar  return luas    No:  def LuasPersegiPanjang(panjang = 2, lebar = None):  luas = panjang\*lebar  return luas |

Pada contoh di atas, kita membuat fungsi luas persegi panjang yang sama seperti sebelumnya. Perhatikan bahwa kita tidak menyertakan anotasi berupa "panjang:int".

Mari kita simpulkan sedikit. 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 (=).

|  |
| --- |
| Yes:  def LuasPersegiPanjang(panjang:int = 2, lebar=None):  pass    No:  def LuasPersegiPanjang(panjang: int=2, lebar = None):  pass |

Pada contoh di atas, kita menggabungkan dalam satu fungsi; parameter panjang menggabungkan anotasi fungsi dan nilai default, sedangkan parameter lebar hanya mendefinisikan nilai default tanpa anotasi fungsi. Perhatikan bahwa spasi ditempatkan pada setiap parameternya.

**Style Guide Prinsip Penamaan pada Python**

Saat membuat variabel, fungsi, hingga kelas, Anda dapat memberikan nama-nama yang beragam. Terkadang, keberagaman tersebut menghasilkan tidak adanya standar dalam kode yang Anda bangun.

Pada materi ini, kita akan belajar beberapa prinsip penamaan saat Anda membangun kode Python. Harapannya, Anda bisa membuat standar nama saat membangun variabel, fungsi, hingga kelas.

Namun, perlu diperhatikan juga bahwa Anda dapat memilih mempertahankan styling yang sudah digunakan sebelumnya untuk menjaga konsistensi internal tim atau perusahaan. Ini karena konsistensi internal lebih diutamakan.

**Catatan**: Pada materi-materi sebelumnya, style guide Python belum diterapkan secara menyeluruh. Sangat disarankan jika Anda mempelajari ulang kode pada materi-materi sebelumnya dengan menerapkan style guide Python.

**Prinsip Overriding**

Nama yang dilihat oleh user publik sebaiknya merefleksikan penggunaan/fungsinya dan bukan implementasinya. Misalnya nama fungsi berikut.

|  |
| --- |
| cariJalan() |

Itu akan lebih mudah dipahami dibanding berikut.

|  |
| --- |
| jalan() |

Algoritma yang digunakan hingga informasi lainnya dari fungsi yang dibangun dapat dijelaskan dalam docstring ataupun komentar.

**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

Satu hal yang perlu diingat ketika Anda membuat sebuah fungsi, sangat tidak disarankan untuk menggunakan frasa atau huruf sebagai awalan fungsi. Awalan fungsi mengacu pada nama fungsi di bagian awal, seperti 'get' pada "get\_name()" atau 'konversi' pada "konversi\_ke\_integer()".

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. Python memiliki prinsip yang berlaku dalam penamaan fungsi atau method sebagai berikut.

1. Atribut dan method name bersifat pre-fixed dengan objek.
2. Function name selalu diawali dengan module name.

Selain penggunaan huruf atau frase 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\_\_.  
     
   Ingat, penamaan ini disebut juga sebagai *dunder* atau *double underscore*oleh programmer Python. Anda sangat tidak disarankan membuat penamaan menggunakan dunder. Misalnya Anda membuat penamaan "\_\_special\_method\_\_", itu sangat tidak disarankan oleh Python karena bisa ada kemungkinan penamaan tersebut telah digunakan oleh Python dan secara tidak sengaja menimpa kode yang sudah ada. Terkecuali penamaan tersebut sudah terdokumentasikan oleh Python seperti '\_\_init\_\_' yang digunakan untuk membuat class constructor.

**Hal-hal yang Harus Dipertimbangkan dalam Penamaan**

Sebelumnya kita membahas detail terkait penamaan sebuah fungsi, method, kelas, hingga hal yang tidak dianjurkan dalam penamaannya. Pembahasan selanjutnya adalah petunjuk untuk mempertimbangkan nama yang tepat. Sekali lagi, penamaan di sini merujuk ke banyak hal, seperti penamaan variabel, fungsi, hingga kelas.

**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. Daripada menggunakan l (huruf l kecil), menggunakan L besar akan sangat membantu.

**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. [ASCII](https://www.cs.cmu.edu/~pattis/15-1XX/common/handouts/ascii.html)adalah sebuah kode karakter yang memetakan set karakter dan umum digunakan dalam angka. Sederhananya, ASCII memetakan karakter-karakter beserta angka yang mewakilinya.

Identifiers merujuk pada nama-nama yang digunakan untuk menyebut variabel, fungsi, kelas, dan kode lainnya dalam Python. Contoh identifiers adalah nama variabel "x", nama fungsi "penjumlahan()", atau nama method "get\_nama()".

**Nama Paket dan Nama Modul**

Masih ingat dengan modul dan package (paket) dalam Python? Modul pada Python adalah file yang berisi kode Python, seperti fungsi, kelas, dan sebagainya. Ketika Anda membuat script atau file Python, hal itu bisa dianggap sebagai modul. Di sisi lain, paket adalah sebuah direktori yang berisi satu atau lebih modul yang terkait dan saling berhubungan.

Penamaan modul sebaiknya pendek atau singkat, menggunakan huruf kecil, dan opsional garis bawah (\_) untuk meningkatkan keterbacaan. Contohnya '\_\_init\_\_.py' atau modul 'math\_operations.py' dengan seluruh kode di dalamnya merupakan fungsi, kelas, method yang berhubungan dengan operasi matematika, seperti penjumlahan, pengurangan, dan sebagainya.

Nama paket juga sebaiknya singkat, menggunakan huruf kecil, dan hindari garis bawah(\_). Contohnya, jika kita membuat paket bernama "math" yang di dalamnya ada modul 'math\_operations.py", pengguna akan memahami bahwa paketnya bernama math yang memiliki banyak modul, seperti salah satunya operasi matematika.

**Nama Kelas**

Saat menamai kelas, gunakan CamelCase atau CapWords. Pastikan semua akronim (misal HTTP) ditulis keseluruhan dengan huruf besar.

**Penulisan Tipe Variabel**

Untuk penamaan variabel, umumnya menggunakan CamelCase atau CapWords, lebih pendek lebih baik.

|  |
| --- |
| T, AnyStr, Num |

Jika terdapat covariant atau contravariant dari sebuah variabel, tambahkan di akhir variabel untuk mempermudah pembacaan. Covariant memungkinkan Anda menggunakan tipe turunan (lebih spesifik) dari yang telah ditentukan sebelumnya. Adapun, contravariant adalah istilah yang merujuk pada kemampuan untuk menggunakan tipe yang lebih umum dari sebelumnya.

|  |
| --- |
| from typing import TypeVar  VT\_co = TypeVar('VT\_co', covariant=True)  KT\_contra = TypeVar('KT\_contra', contravariant=True) |

**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. Contoh kodenya sebagai berikut.

|  |
| --- |
| class DivideByZeroError(Exception):  def \_\_init\_\_(self, message="Division by zero is not allowed"):  super().\_\_init\_\_(message)  def divide\_numbers(a, b):  if b == 0:  raise DivideByZeroError("Cannot divide by zero")  return a / b  try:  result = divide\_numbers(10, 0)  except DivideByZeroError as e:  print(f"Error: {e}")  """  Output:  Error: Cannot divide by zero  """ |

Pada contoh di atas, kita membuat sebuah pengecualian berasal dari kelas yang dibuat. Kita membuat kelas bernama "DivideByZeroError" yang menginduk pada kelas Exception dari Python. Perhatikan bahwa kita menempatkan kata error di akhir penamaannya.

**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.

**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.

**Argumen Fungsi dan Method**

Dalam pembuatan fungsi dan method pada suatu kelas, ada beberapa hal yang perlu dipertimbangkan.

* Gunakan self sebagai argumen pertama jika Anda membuat instance method.
* Gunakan cls sebagai argumen pertama ketika Anda membuat class method.

Jika nama argumen fungsi adalah *reserved keyword*, tambahkan garis bawah di akhir nama argumen. Jangan mengorbankan keterbacaan nama dengan menyingkatnya. Mengganti argumen bernama class dengan class\_ atau kelas, lebih baik daripada clss.

**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.

Untuk menghindari kesamaan dengan subkelas, gunakan \_\_dimulai\_dua\_garis\_nama\_method untuk memanggil proses yang tepat. Python menggabungkan nama modul dengan nama kelas. Misal ada suatu kelas bernama Foo, jika kelas Foo memiliki atribut \_\_a, kita tidak dapat mengaksesnya melalui Foo.\_\_a, tetapi Foo.\_Foo\_\_a. Mulai dengan dua garis bawah hanya digunakan jika terjadi konflik dengan atribut di kelas atau subkelas lainnya.

**Konstanta**

Dalam memberikan nama variabel bertipe konstanta, umumnya didefinisikan pada bagian atas modul dengan huruf besar semuanya, misalnya 'PI = 3,14'  atau  'TOTAL = 4.14213'.

**Selalu Persiapkan untuk Inheritance**

Saat membangun metode dan variabel dalam sebuah kelas, sebaiknya Anda dapat langsung mengetahui atribut pada metode dan variabel tersebut, entah 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 bersifat non-publik adalah suatu variabel atau method yang hanya digunakan untuk lingkup tertentu dan tidak diakses secara langsung di luar. Contohnya berikut.

|  |
| --- |
| class MyClass:  def \_\_init\_\_(self):  self.\_private\_var = 42 # Variabel non publik dengan awalan satu garis bawah  self.\_secret\_list = [1, 2, 3] # Variabel non publik lainnya  def \_private\_method(self):  print("Ini adalah method non publik")  def public\_method(self):  print("Ini adalah method publik")  self.\_private\_method() # Method publik dapat memanggil method non publik |

Pada contoh di atas, method '\_private\_method' merupakan jenis fungsi yang tidak diakses secara langsung. Anda bisa melihat pada method 'public\_method', tempat kita menggunakan method private di sana. Selain itu, variabel seperti '\_private\_var' atau '\_secret\_list' merupakan variabel non\_publik yang tidak digunakan secara langsung ketika kelas dipanggil.

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.

Kategori lain dari atribut adalah "subclass API", umumnya disebut protected pada bahasa lain. Sebuah kelas dapat didesain untuk diwariskan (inherited-from), misalnya untuk memodifikasi atau menjadi ekstensi dari perilaku (behavior) kelas. Dalam mendesain kelas-kelas sejenis, pastikan untuk membuat keputusan eksplisit, variabel/method yang memiliki atribut publik, bagian dari subclass API, dan yang hanya anda gunakan secara internal.

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 yang 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.

Sekali lagi, semua materi style guide kali ini mengacu pada PEP8 yang dapat Anda baca lebih lanjut dalam link [berikut](https://peps.python.org/pep-0008/" \t "_blank).