README.md

ingkasan :

- Bab 9 *Kelas

Kelas menyediakan sarana bundling data dan fungsionalitas bersama. Membuat kelas baru akan membuat tipe objek baru, yang memungkinkan instance baru dari tipe tersebut dibuat. Setiap instance kelas dapat memiliki atribut yang melekat padanya untuk mempertahankan statusnya. Ini adalah campuran dari mekanisme kelas yang ditemukan di C++ dan Modula-3. Kelas Python menyediakan semua fitur standar Pemrograman Berorientasi Objek: mekanisme pewarisan kelas memungkinkan banyak kelas dasar, kelas turunan dapat mengganti metode apa pun dari kelas atau kelas dasarnya, dan metode dapat memanggil metode kelas dasar dengan nama yang sama . Seperti di Modula-3, tidak ada singkatan untuk mereferensikan anggota objek dari metodenya: fungsi metode dideklarasikan dengan argumen pertama yang eksplisit yang mewakili objek, yang disediakan secara implisit oleh panggilan. Seperti di Smalltalk,

*Sebuah Kata Tentang Nama dan Objek

Objek memiliki individualitas, dan banyak nama dapat diikat ke objek yang sama. Ini biasanya tidak dihargai pada pandangan pertama di Python, dan dapat diabaikan dengan aman saat berhadapan dengan tipe dasar yang tidak dapat diubah. Ini biasanya digunakan untuk kepentingan program, karena alias berperilaku seperti penunjuk dalam beberapa hal.

*Cakupan Python dan Ruang Nama

Definisi kelas memainkan beberapa trik rapi dengan ruang nama, dan Anda perlu mengetahui cara kerja ruang lingkup dan ruang nama untuk memahami sepenuhnya apa yang terjadi. Kebetulan, pengetahuan tentang subjek ini berguna untuk semua programmer Python tingkat lanjut. Mari kita mulai dengan beberapa definisi. Namespace adalah pemetaan dari nama ke objek.

Sebagian besar ruang nama saat ini diimplementasikan sebagai kamus Python, tapi itu biasanya tidak terlihat sama sekali, dan mungkin berubah di masa mendatang. Dalam arti himpunan atribut suatu objek juga membentuk namespace. Funcname, modname adalah objek modul dan funcname adalah atributnya. The_answer akan menghapus atribut the_answer dari objek yang dinamai dengan modname.

Ruang nama dibuat pada momen yang berbeda dan memiliki masa hidup yang berbeda. Ruang nama yang berisi nama bawaan dibuat saat juru bahasa Python dijalankan, dan tidak pernah dihapus. Pernyataan yang dieksekusi oleh pemanggilan tingkat atas dari juru bahasa, baik dibaca dari file skrip atau secara interaktif, dianggap sebagai bagian dari modul bernama **main**, sehingga mereka memiliki namespace globalnya sendiri. Ruang nama lokal untuk suatu fungsi dibuat saat fungsi dipanggil, dan dihapus saat fungsi mengembalikan atau memunculkan pengecualian yang tidak ditangani dalam fungsi.

Tentu saja, pemanggilan rekursif masing-masing memiliki namespace lokalnya sendiri. Lingkup adalah wilayah tekstual dari program Python di mana namespace dapat diakses secara langsung. «Dapat diakses langsung» di sini berarti referensi yang tidak memenuhi syarat ke sebuah nama mencoba menemukan nama tersebut di namespace. Meskipun cakupan ditentukan secara statis, cakupan digunakan secara dinamis.

Jika sebuah nama dideklarasikan secara global, maka semua referensi dan penetapan langsung menuju ke lingkup berikutnya hingga terakhir yang berisi nama global modul. Biasanya, cakupan lokal mereferensikan nama lokal dari fungsi saat ini. Definisi kelas menempatkan namespace lain dalam lingkup lokal. Keunikan khusus Python adalah - jika tidak ada pernyataan global atau nonlokal yang berlaku - penugasan ke nama selalu masuk ke ruang lingkup terdalam.

*Contoh Cakupan dan Ruang Nama

Penetapan nonlokal mengubah pengikatan spam scope_test, dan penetapan global mengubah pengikatan tingkat modul.

*Pandangan Pertama tentang Kelas Kelas memperkenalkan sedikit sintaks baru, tiga tipe objek baru, dan beberapa semantik baru.

*Sintaks Definisi Kelas Definisi

kelas, seperti definisi fungsi harus dijalankan sebelum memiliki efek apa pun. Secara khusus, definisi fungsi mengikat nama fungsi baru di sini. Ketika definisi kelas dibiarkan normal, objek kelas dibuat.

*Objek Kelas

Instansiasi kelas menggunakan notasi fungsi. Anggap saja objek kelas adalah fungsi tanpa parameter yang mengembalikan instance baru dari kelas. Membuat instance baru dari kelas dan menugaskan objek ini ke variabel lokal x. Operasi instantiasi membuat objek kosong.

Ketika kelas mendefinisikan metode **init**, instantiasi kelas secara otomatis memanggil **init** untuk instance kelas yang baru dibuat. Tentu saja, metode **init** mungkin memiliki argumen untuk fleksibilitas yang lebih besar. Dalam hal ini, argumen yang diberikan kepada operator instansiasi kelas diteruskan ke **init**.

*Objek Instance

Satu-satunya operasi yang dipahami oleh objek instan adalah referensi atribut. Jenis lain dari referensi atribut instance adalah metode. Metode adalah fungsi yang «milik» suatu objek. Nama metode yang valid dari objek instance bergantung pada kelasnya.

F adalah referensi metode yang valid, karena MyClass. Saya tidak, sejak MyClass. F tidak sama dengan MyClass.

*Objek Metode

F adalah objek metode, dan dapat disimpan dan dipanggil di lain waktu. F dipanggil tanpa argumen di atas, meskipun definisi fungsi untuk f menentukan argumen. Secara umum, memanggil metode dengan daftar n argumen sama dengan memanggil fungsi yang sesuai dengan daftar argumen yang dibuat dengan menyisipkan objek instance metode sebelum argumen pertama. Ketika objek metode dipanggil dengan daftar argumen, daftar argumen baru dibuat dari objek instance dan daftar argumen, dan objek fungsi dipanggil dengan daftar argumen baru ini.

*Variabel Kelas dan Instance

Seperti yang dibahas dalam A Word About Names and Objects, data yang dibagikan mungkin memiliki efek mengejutkan dengan melibatkan objek yang dapat berubah seperti daftar dan kamus.

*Catatan Acak

Atribut data dapat direferensikan oleh metode maupun oleh pengguna biasa dari suatu objek. Dengan kata lain, kelas tidak dapat digunakan untuk mengimplementasikan tipe data abstrak murni. Nyatanya, tidak ada apa pun dalam Python yang memungkinkan untuk memaksakan penyembunyian data — semuanya didasarkan pada konvensi. Klien harus menggunakan atribut data dengan hati-hati — klien dapat mengacaukan invarian yang dikelola oleh metode dengan menandai atribut data mereka. Saya menemukan bahwa ini benar-benar meningkatkan keterbacaan metode: tidak ada kemungkinan membingungkan variabel lokal dan variabel instan ketika melihat sekilas melalui metode. Seringkali, argumen pertama dari sebuah metode disebut self. Ini tidak lebih dari sebuah konvensi: nama self sama sekali tidak memiliki arti khusus untuk Python. Sekarang f, g dan h adalah semua atribut kelas C yang mengacu pada objek fungsi, dan akibatnya mereka semua adalah metode turunan dari C — h sama persis dengan g. Perhatikan bahwa praktik ini biasanya hanya berfungsi untuk membingungkan pembaca suatu program. Metode dapat mereferensikan nama global dengan cara yang sama seperti fungsi biasa.

*Pewarisan

Tentu saja, sebuah fitur bahasa tidak akan layak disebut «kelas» tanpa mendukung pewarisan. Kelas turunan dapat menggantikan metode dari kelas dasarnya. Karena metode tidak memiliki hak istimewa saat memanggil metode lain dari objek yang sama, metode dari kelas dasar yang memanggil metode lain yang didefinisikan dalam kelas dasar yang sama mungkin akan memanggil metode dari kelas turunan yang menimpanya. Metode utama dalam kelas turunan mungkin sebenarnya ingin memperluas daripada sekadar mengganti metode kelas dasar dengan nama yang sama.

*Pewarisan Ganda

Pendekatan ini dikenal dalam beberapa bahasa pewarisan ganda lainnya sebagai metode panggilan-berikutnya dan lebih kuat daripada panggilan super yang ditemukan dalam bahasa pewarisan tunggal. Pengurutan dinamis diperlukan karena semua kasus pewarisan berganda menunjukkan satu atau lebih hubungan intan. Misalnya, semua kelas mewarisi dari objek, jadi setiap kasus pewarisan berganda menyediakan lebih dari satu jalur untuk mencapai objek.

*Variabel Pribadi

Variabel instan «Pribadi» yang tidak dapat diakses kecuali dari dalam objek tidak ada di Python. Karena ada use-case yang valid untuk class-private member , ada dukungan terbatas untuk mekanisme seperti itu, yang disebut name mangling. Pengidentifikasi apa pun dalam bentuk __spam secara tekstual diganti dengan _classname__spam, di mana classname adalah nama kelas saat ini dengan garis bawah di depan dihilangkan. Penguraian ini dilakukan tanpa memperhatikan posisi sintaksis dari pengidentifikasi, asalkan terjadi dalam definisi suatu kelas.

Contoh di atas akan berfungsi bahkan jika MappingSubclass akan memperkenalkan pengenal __update karena diganti dengan _Mapping__update di kelas Mapping dan _MappingSubclass__update di kelas MappingSubclass.

*Odds and Ends

Sepotong kode Python yang mengharapkan tipe data abstrak tertentu sering kali dapat diteruskan ke kelas yang meniru metode tipe data tersebut. Misalnya, jika Anda memiliki fungsi yang memformat beberapa data dari objek file, Anda dapat menentukan kelas dengan metode baca dan baca yang mendapatkan data dari buffer string, dan meneruskannya sebagai argumen.

*Iterator

Fungsi mengembalikan objek iterator yang mendefinisikan metode **next** yang mengakses elemen dalam wadah satu per satu. Setelah melihat mekanisme di balik protokol iterator, mudah untuk menambahkan perilaku iterator ke kelas Anda. Tentukan metode **iter** yang mengembalikan objek dengan metode **next**.

*Generator

Apapun yang bisa dilakukan dengan generator juga bisa dilakukan dengan iterator berbasis kelas seperti yang dijelaskan di bagian sebelumnya. Apa yang membuat generator begitu kompak adalah bahwa metode **iter** dan **next** dibuat secara otomatis. Fitur utama lainnya adalah variabel lokal dan status eksekusi disimpan secara otomatis di antara panggilan. Selain pembuatan metode otomatis dan menyimpan status program, ketika generator berhenti, mereka secara otomatis menaikkan StopIteration.

*Ekspresi Generator

Ekspresi ini dirancang untuk situasi di mana generator digunakan langsung oleh fungsi penutup. Ekspresi generator lebih kompak tetapi kurang fleksibel daripada definisi generator lengkap dan cenderung lebih ramah memori daripada pemahaman daftar yang setara.