**DAFTAR ISI**

**Latar Belakang 4**

**Rumusan Masalah 5**

**Tujuan Penulisan 5**

**Pengertian Bahasa Pemrograman 6**

**Sejarah Bahasa Pemrograman 6**

**Pemrograman Python 12**

**Pengertian Syntax dan Semantik 13**

**BAB I Describing Syntax and Semantics 15**

1. **Comments 16**
	1. **Comment untuk satu baris 16**
	2. **Comment untuk beberapa baris 16**
2. **Whitespace 16**
3. **Indentation 16**
4. **Tokens 17**
	1. **Keyword 17**
	2. **Operator 17**
	3. **Punctuator 19**
	4. **Delimiter 19**
	5. **Literal 19**
	6. **Print 20**
	7. **Raw Input 20**
	8. **If, elif, else (conditional) 20**
	9. **True, False, NOT (ekspresi Boolean) 21**
5. **Expressions 21**
6. **Interpreter Errors 21**

**BAB 2 Name, Binding, and Scope 23**

1. **Nama Variabel 24**
2. **Tipe dan Jenis Data 24**
	1. **Tipe Data Bilangan 25**
	2. **Tipe Data Teks 25**
	3. **Tipe Data Boolean 25**
3. **Binding 25**
	1. **Bindin Type 25**
	2. **Storage Binding and Allocation 25**
4. **Variabel Scope 26**

**BAB 3 Data Types 27**

1. **Primitive Data Type 28**
2. **Character String Type 28**
3. **User Defined Ordinal 29**
4. **Array Type 29**
	1. **Heterogeneous Arrays 30**
	2. **Associative Arrays 30**
5. **Record Types 30**
6. **Tuple Types 30**
7. **List Type 30**
8. **Union Type 31**
9. **Pointer and Reference Types 31**
10. **Type Checking 31**
11. **Strong Typing 31**

**BAB 4 Espression and Assignment Statement 32**

1. **Operator Aritmatika 33**
2. **Overloading dan Type Conversion 34**
	1. **List 34**
3. **Relation Expression and Boolean 38**
4. **Short Circuit Evaluation Of Logical Expressions 39**
5. **Assignment Statement 40**

**BAB 5 Control Structure And Statement 41**

1. **Statement-Level Control Structures 42**
	1. **Control Structure 42**
	2. **Selection Statements 42**
	3. **Two Way Selection Statements 42**
2. **Multiple Way Selection Statements Using Switch Case 42**
	1. **If Statements 43**
	2. **If else 43**
	3. **Elif 44**
	4. **Python Break 45**
	5. **Python Continue 46**
3. **Clause Form 47**
	1. **If Statement 47**
	2. **While Statement 47**
	3. **For Statement 48**
4. **Try Statement 49**
	1. **Menangani Error Pembacaan File 49**
5. **With Statement 51**
6. **Iterative Statements 52**
7. **Counter Controlled Loops 52**
8. **Logically Controlled Loops 53**
	1. **For Loop 53**
	2. **While Loop 53**
9. **User Loaded Loop Control Mechanism 54**
	1. **Sistem Kontrol Loop Terbuka 54**
	2. **Sistem Kontrol Loop Tertutup 54**

**BAB 6 Subprogram 55**

1. **Prosedure Dalam Subprogram 56**
2. **Fungsi Dalam Subprogram 57**
3. **Design Issue of Subprograms 60**
4. **Local Referencing Environments 60**
5. **Parameter Passing Methods 62**
6. **Generic Subprogram 65**
7. **Polymorphism(Overloading) 65**

**BAB 7 Abstract Data Type 68**

1. **Introduction And Concept of Abstraction 69**
2. **Parameterized Abstract Data Type 70**
3. **Naming Encapsulation 71**
4. **Encapsulation Construct 72**
5. **Example of Abstraction 73**

**BAB 8 Object-Oriented Programming 75**

1. **Pengertian OOP 76**
2. **Konsep OOP 76**
3. **Karakteristik OOP 77**
4. **Semuanya Merupaka Objek 78**
5. **Semuanya Memiliki Kelas?.. 79**
6. **Penggunaan Self 80**
7. **Penggunaan \_init\_ 81**
8. **Instance Attributes and Methods 83**
9. **Static Methods 83**
10. **Class Methods 84**
11. **Inheritance 85**

**BAB 9 Concurrency 88**

1. **Fungsi Matematika 89**
2. **Higher-Order Function 89**
3. **Purity 90**
4. **Perbedaan Functional dan Imperative Language 90**

**BAB 10 Exception Handling And Event Handling 91**

1. **Exeption Handling 92**
2. **Syntax Errors 94**
3. **Exeptions 94**
4. **Raising Exceptions 98**
5. **User-defined Exeptions 99**
6. **Defining Clean-up Actions 100**
7. **Predefined Clean-up Actions 101**

**BAB 11 Logic Programming Languages 103**

1. **Penjelasan Singkat Mengenai Predicate Calculus 104**
2. **Preposisi 104**
3. **Overview of Logic Programming 106**
4. **Terms 107**
5. **Fact Statements 108**
6. **Rule Statements 108**
7. **Goal Statements 108**
8. **The Inferencing Process of Prolog 109**
9. **Simple Arithmetic 109**
10. **List Structures 109**
11. **Latar Belakang**

Bahasa pemrograman atau sering diistilahkan juga dengan Bahasa komputer adalah teknik komando/instruksi untuk memerintahkan komputer. Bahasa pemrograman ini merupakan suatu himpunan dari aturan sintaks dan semantik yang dipakai untuk mendefinisikan program komputer. Bahasa ini memungkinkan seorang programmer dapat menentukan secara persis data mana yang akan diolah oleh komputer, bagaimana data ini akan disimpan/diteruskan, dan jenis langkah apa secara persis yang akan diambil dalam berbagai situasi. Menurut tingkat kedekatannya dengan mesin komputer bahasa pemrograman terdiri dari: Bahasa Mesin yaitu memberikan perintah kepada komputer dengan memakai kode bahasa biner, contohnya 01000100101010. Bahasa Tingkat Rendah atau dikenal dengan istilah bahsa rakitan(bahasa.inggris Assembly), yaitu memberikan perintah kepada komputer dengan memakai kode-kode singkat(kode mnemonic), contohnya MOV, LOOP. Bahasa Tingkat Menengah yaitu bahasa komputer yang memakai campuran instruksi dalam kata-kata bahasa manusia. Bahasa Tingkat Tinggi, yaitu Bahasa komputer yang memakai instruksi berasal dari unsur kata-kata Bahasa manusia, contohnya begin, end, if, for, while, and, or. Sebagaian besar bahasa pemrograman digolongkan sebagai Bahasa Tingkat Tinggi, hanya bahasa C yang digolongkan sebagai Bahasa Tingkat Menengah dan Assembly yang merupakan Bahasa Tingkat Rendah.

Peran bahasa pemrograman dalam membuat aplikasi berperan sangat besar. Oleh karena itu, bahasa pemrograman sebagai instruksi untuk memerintah komputer yang akan menghubungkan antara pembuat program dengan aplikasi yang akan dibuat atau dijalankanya.

Dengan banyaknya bahasa pemrograman yang ada saat ini, seperti bahasa pemrograman C, C++, python, perl, java dan ruby yang digunakan oleh perusahaan, maka bahasa pemrograman yang dinamis lebih banyak digunakan oleh para programmer dalam menjalankan aktifitasnya Bahasa pemrograman berorientasi objek merupakan tujuan yang paling utama. Pentingnya bahasa pemrograman berorientasi objek sebagai media untuk membuat aplikasi yang akan dibuat seperti aplikasi website yang pada akhirnya akan berdampak positif pada seorang programer, tidak saja pada developer website itu sendiri yang mempunyai tujuan untuk mendapatkan keuntungan, tetapi juga bermanfaat positif bagi masyarakat sebagai pengguna. Oleh karena itu bahasa pemrograman pada akhirnya harus mempunyai efisiensi yang baik dalam bahasa pemrogramanya, terutama dalam hal efisiensi coding

1. **Rumusan Masalah**

Agar lebih memahami dan menguasai tentang pembelajaran Programming Language Concepts

1. **Tujuan Penulisan**

Tujuan kami menulis Laporan ini adalah untuk menyelesaikan tugas akhir Programming language Concepts dan mendapatkan nilai yang bagus.

1. **Pengertian Bahasa Pemrograman**

Bahasa pemograman yang disebut juga bahasa komputer atau bahasa pemograman komputer, adalah instruksi standar untuk memerintah komputer. Bahasa pemrograman adalah suatu himpunan dari aturan sintaks dan semantik yang dipakai untuk mendefinisikan program pada komputer. Bahasa pemrograman ini memungkinkan seseorang untuk dapat menentukan secara persis data mana yang akan diolah oleh komputer, bagaimana datanya akan disimpan atau diteruskan,dan jenis langkah yg akan diambil dalam berbagai situasi.

Bahasa komputer menurut tingkat kedekatannya dengan mesin terdiri dari bahasa mesin ( memakai kode bahasa biner), bahasa tingkat rendah(memakai kode-kode singkat/ kode mnemonic), bahasa tingkat menengah(memakai campuran instruksi dalam kata2 bahasa manusia dan instruksi yang bersifat simbolik), dan bahasa tingkat tinggi (memakai instruksi berasal dari unsur kata2 bahasa manusia).

1. **Sejarah Bahasa Pemrograman**

Sejak pertama komputer difference engine diciptakan oleh Charles Babbage pada tahun 1822, komputer membutuhkan sejumlah instruksi untuk melakukan suatu tugas tertentu. Instruksi-instruksi ini dikenal sebagai bahasa pemrograman. Bahasa komputer mulanya terdiri dari sejumlah langkah pengkabelan untuk membuat suatu program; hal ini dapat dipahami sebagai suatu rangkaian pengetikan kedalam komputer dan kemudian dijalankan.

Pada awalnya, difference engine-nya Charles Babbage hanya dibuat untuk menjalankan tugas dengan menggunakan perpindahan gigi roda untuk menjalankan fungsi kalkukasi. Jadi, bentuk awal dari bahasa komputer adalah berupa gerakan secara mekanik, selanjutnya gerakan mekanik tersebut digantikan dengan sinyal listrik ketika pemerintah AS mengembangkan ENIAC pada tahun 1942, tetapi masih banyak mengadopsi prinsip-prinsip dasar dari Babbage’s engine yang mana diprogram dengan mengeset switch dan perkabelan pada seluruh sistem pada setiap “program” maupun kalkulasi.

Pada 1945, John Von Neumann yang bekerja pada Institute for Advanced Study mengemukakan dua konsep yang secara langsung mempengaruhi masa depan dari bahasa pemrograman komputer. Yang pertama dikenal sebagai “shared-program technique”. Pada teknik ini dinyatakan bahwa hardware komputer haruslah sederhana dan tidak perlu dilakukan pengkabelan dengan menggunakan tangan untuk setiap program. Sebagai gantinya, instruksi-instruksi yang lebih kompleks harus digunakan untuk mengendalikan perangkat keras yang lebih sederhana, hal ini memungkinkan komputer diprogram ulang dengan cepat. Konsep yang kedua, Von Neumann menyebutnya sebagai “conditional control transfer”. Ide ini berkembang menjadi bentuk subrutin, atau blok kode yang kecil yang dapat panggil berdasarkan aturan tertentu, dari pada suatu himpunan tunggal urutan kronologis yang harus dijalankan oleh komputer. Bagian kedua dari ide tersebut menyatakan bahwa kode komputer harus dapat bercabang berdasarkan pernyataan logika seperti IF (ekspresi) THEN, dan perulangan seperti FOR statement. “Conditional control transfer” mengembangkan ide adanya “libraries,” yang mana merupakan blok kode yang dapat digunakan berulang kali.

Pada 1949, setelah beberapa tahun Von Neumann bekerja, bahasa Short Code dilahirkan, yang merupakan bahasa komputer yang pertama untuk peralatan elektronik yang membutuhkan programmer untuk mengubah perintah kedalam 0 dan 1 dengan tangan.

Pada 1957, bahasa khusus yang pertama muncul dalam bentuk FORTRAN yang merupakan singkatan dari sistem FORmula TRANslating. Bahasa ini dirancang pada IBM untuk perhitungan scientific. Komponen-komponennya sangat sederhana, dan menyediakan bagi programmer akses tingkat rendah kedalam komputer. Sampai saat ini, bahasa ini terbatas pada hanya terdiri dari perintah IF, DO, dan GOTO, tetapi pada waktu itu, perintah-perintah ini merupakan lompatan besar kearah depan. Type data dasar yang digunakan sampai sekarang ini dimulai dari FORTRAN, hal ini meliputi variabel logika (TRUE atau FALSE), dan bilangan integer, real, serta double-precision.

FORTRAN sangat baik dalam menangani angka-angka, tetapi tidak terlalu baik untuk menangani proses input dan output, yang mana merupakan hal yang penting pada komputasi bisnis. Komputasi bisnis mulai tinggal landas pada 1959, dengan dikembangkannya COBOL, yang dirancang dari awal sebagai bahasa untuk para pebisnis. Type data yang ada hanya berupa number dan text string. Hal tersebut juga memungkinkan pengelompokan menjadi array dan record, sehingga data di telusuri dan diorganisasikan dengan lebih baik. Sesuatu hal yang menarik untuk dicatat bahwa suatu program COBOL dibuat menyerupai suatu essay, dengan empat atau lima bagian utama yang membentuk keseluruhan yang tertata dengan baik. Perintah-perintah COBOL sangat menyerupai tata bahasa English, sehingga membuatnya agak mudah dipelajari. Semua ciri-ciri ini dikembangkan agar mudah dipelajari dan mudah diterapkan pada dunia bisnis.

Pada 1958, John McCarthy di MIT membuat bahasa LISt Processing (atau LISP), yang dirancang untuk riset Artificial Intelligence (AI). Karena dirancang untuk fungsi spesialisasi yang tinggi, maka tata cara penulisannya jaring kelihatan sebelum ataupun sesudahnya. Sesuatu perbedaan yang paling nyata dari bahasa ini dengan bahasa lain adalah dasar dan type satu-satunya adalah list, yang ditandai dengan suatu urutan item yang dicakup dengan tanda kurung. Program LISP sendirinya dibuat sebagai suatu himpunan dari list, sehingga LISP memiliki kemampuan yang khusus untuk memodifikasi dirinya, dan juga dapat berkembang sendiri. Tata cara penulisan LISP dikenal sebagai “Cambridge Polish,” sebagaimana dia sangat berbeda dari logika Boolean (Wexelblat, 177).

Bahasa Algol dibuat oleh suatu komite untuk pemakaian scientific pada tahun 1958. Kontribusi utamanya adalah merupakan akar dari tiga bahasa selanjutnya yaitu Pascal, C, C++, dan Java. Dia juga merupakan bahasa pertama dengan suatu tata bahasa formal, yang dikenal sebagai Backus-Naar Form atau BNF (*McGraw-Hill Encyclopedia of Science and Technology*, 454).

Pascal dimulai pada tahun 1968 oleh Niklaus Wirth. Tujuan pengembangannya adalah untuk kebutuhan pengajaran. Pada awalnya bahasa ini dikembangkan bukan dengan harapan adopsi pemakaian secara luas. Prinsipnya mereka mengembangkannya untuk alat pengajaran pemrograman yang baik seperti kemampuan debug dan perbaikan sistem dan dukungan kepada mikroprosesor komputer yang digunakan pada institusi pendidikan. Pascal dirancang dengan pendekatan yang sangat teratur (terstruktur), dia mengkombinasikan kemampuan yang terbaik dari bahasa-bahasa saat itu, COBOL, FORTRAN, dan ALGOL. Dalam pengerjaannya banyak perintah-perintah yang tidak teratur dan aneh dihilangkan, sehingga sangat menarik bagi pemakai. Pascal juga mengembangkan variabel dinamis, dimana variabel dapat dibuat ketika suatu program sedang berjalan, melalui perintah NEW dan DISPOSE. Tetapi Pascal tidak mengimplementasikan suatu array dinamis, atau kelompok dari variabel-variabel, yang mana sangat dibutuhkan, dan merupakan salah satu penyebab kekalahannya.

C dikembangkan pada tahun 1972 oleh Dennis Richie ketika sedang bekerja pada Bell Labs di New Jersey. Transisi pemakaian dari bahasa umum yang pertama ke bahasa umum sampai hari ini yaitu transisi antara Pascal dan C, C merupakan perkembangan dari B dan BCPL, tetapi agak menyerupai Pascal. Semua fasilitas di Pascal, termasuk perintah CASE tersedia di C. C menggunakan pointer secara luas dan dibangun untuk kecepatan dengan kelemahannya yaitu menjadi sulit untuk dibaca. Tetapi karena dia menghilangkan semua kelemahan yang terdapat di Pascal, sehingga dengan cepat mengambil alih posisi Pascal. Ritchie mengembangan C untuk sistem Unix yang baru pada saat yang bersamaan. Oleh karena ini, C dan Unix saling berkaitan. Unix memberikan C beberapa fasilitas besar seperti variabel dinamis, multitasking, penanganan interrupt, forking, dan strong low-level,input-output. Oleh karena itu, C sangat sering digunakan untuk pemrograman sistem operasi seperti Unix, Windows, MacOS, dan Linux.

Pada akhir tahun 1970 dan awal 1980, suatu metode pemrograman yang baru telah dikembangkan. Ha tersebut dikenal sebagai Object Oriented Programming, atau OOP. Object merupakan suatu potongan dari data yang dapat dipaket dan dimanipulasi oleh programmer. Bjarne Stroustroup menyukai metode ini dan mengembangkan lanjutan dari C yang dikenal sebagai “C With Classes.” Kemampuan lanjutan ini dikembangkan menjadi bahasa C++ yang diluncurkan pada tahun 1983. C++ dirancang untuk mengorganisasikan kemampuan dasar dari C dengan OOP, dengan tetap mempertahankan kecepatan dari C dan dapat dijalankan pada komputer yang tipe berlainan. C++ sering kali digunakan dalam simulasi, seperti game. C++ menyediakan cara yang baik untuk memanipulasi ratusan instance dari manusia didalan elevator, atau pasukan yang diisi dengan tipe prajurit yang berbeda. Bahasa ini menjadi pilihan pada mata kuliah AP Computer Science sampai hari ini.

Pada awal 1990′s, interaktif TV adalah teknologi masa depan. Sun Microsystems memutuskan bahwa interaktif TV membutuhkan suatu hal yang khusus, yaitu bahasa portable (bahasa yang dapat berjalan pada banyak jenis mesin yang berbeda). Bahasa ini dikenal sebagai Java. Pada tahun 1994, team proyek Java mengubah fokus mereka ke web, yang mana berubah menjadi sesuatu yang menjanjikan setelah interactive TV gagal. Pada tahun berikutnya, Netscape menyetujui pemakaian Java pada internet browser mereka, Navigator. Sampai titik ini, Java menjadi bahasa masa depan.

Java mempunyai tujuan yang besar dan merupakan bahasa yang baik menurut buku text, pada kenyataanya “bahasa tersebut tidak”. Dia memiliki masalah yang serius dalam optimasi, dengan arti program yang ditulis dengannya berjalan dengan lambat. Dan Sun telah membuat cacat penerimaan terhadap Java dengan pertikaian politis dengan Microsoft. Tetapi Java telah dinyatakan sebagai bahasa untuk instruksi masa depan dan benar-benar menerapkan object-oriented dan teknik tingkat tinggi seperti kode yang portable dan garbage collection.

Visual Basic sering diajari sebagai bahasa pemrograman dasar yang mengacu pada bahasa BASIC yang dikembangkan pada tahun 1964 oleh John Kemeny dan Thomas Kurtz. BASIC adalah bahasa yang sangat terbatas dan dirancang untuk orang yang bukan computer science. Perintah-perintah dijalankan secara berurutan, tetapi kendali program dapat berubah berdasarkan IF..THEN, dan GOSUB yang mana menjalankan suatu blok kode dan kembali ketitik semula didalam alur program. Microsoft telah mengembangkan BASIC ke dalam produk Visual Basic (VB). Jantung dari VB adalah form, atau suatu window kosos dimana anda dapat drag dan drop komponen seperti menu, gambarm dan slider bars. Item-item ini dikenal sebagai “widgets.” Widget memiliki properti (seperti warna) dan events (seperti klik dan double klik) dan menjadi pusat dari pengembangan antarmuka dengan pemakai diberbagai bahasa program dewasa ini. VB merupakan program yang banyak digunakan untuk membuat interface sederhana ke produk Microsoft lainnya seperti Excel dan Access tanpa membaca banyak kode, dengannya dapat dimungkinkan untuk dibuat aplikasi yang lengkap.

Perl telah sering digambarkan sebagai “duct tape of the Internet,” karena sering digunakan sebagai engine untuk interface web atau pada script untuk memodifikasi file konfigurasi. Dia memiliki fungsi text matching yang sangat baik sehingga membuatnya menjadi hal yang ideal untuk pekerjaan tersebut. Perl dikembangkan oleh Larry Wall pada 1987 karena fasilitas pada sed dan awk pada Unix (digunakan untuk manipulasi text) tidak mencukupi kebutuhannya. Tergantung kepada siapa anda bertanya, Perl adalah singkatan dari Practical Extraction and Reporting Language atau Pathologically Eclectic Rubbish Lister.

Bahasa pemrograman telah berkembangan dari masa kemasa dan tetap dikembangkan dimasa depan. Mereka dimulai dari suatu daftar langkap pengkabelan agar komputer menjalankan tugas tertentu. Langkah-langkah ini berkembang menjadi software dan memiliki kemampuan yang lebih baik. Bahasa umum yang pertama menekankan pada kesederhanaan dan untuk satu tujuan saja, sedangkan bahasa dewasa ini terbagi atas bagaimana mereka diprogram, sehingga mereka dapat digunakan untuk semua tujuan. Dan mungkin bahasa yang akan datang lebih natural dengan penemuan pada quantum dan komputer-komputer biologis.

1. **Pemrograman Python**

Bahasa pemrograman pada dasarnya merupakan instruksi standar untuk memerintahkan [komputer](http://id.wikipedia.org/wiki/Komputer) untuk melakukan sesuatu. Bahasa pemrograman ini merupakan suatu himpunan dari aturan [sintaks](http://id.wikipedia.org/wiki/Sintaks) dan [semantik](http://id.wikipedia.org/wiki/Semantik)yang dipakai untuk mendefinisikan [program komputer](http://id.wikipedia.org/wiki/Program_komputer). Bahasa ini juga memungkinkan seorang programmer dapat menentukan secara persis data mana yang akan diolah oleh komputer, bagaimana data ini akan disimpan/diteruskan, dan jenis [langkah](http://id.wikipedia.org/wiki/Algoritma) apa yang akan diambil dalam berbagai situasi. Diberbagai belahan dunia terdapat banyak sekali Bahasa pemrograman yang ada. Bahasa pemrograman pada umumnya dikelompokkan berdasarkan tingkat kedekatannya dengan mesin, yakni Bahasa mesin, Bahasa tingkat rendah, Bahasa tingkat menengah dan juga Bahasa tingkat tinggi. Bahasa mesin pada umumnya memberikan perintah kepada komputer dengan menggunakan kode Bahasa biner. Bahasa tingkat rendah, dapat juga disebut dengan Bahasa rakitan atau *assembly* yang merupakan Bahasa yang paling dekat dengan Bahasa mesin dan biasanya digunakan kode-kode singkat untuk memberikan perintah. Bahasa tingkat menengah merupakan Bahasa yang dapat dikelompokkan ke dalam Bahasa tingkat rendah dan juga dapat dikelompokkan ke dalam Bahasa tingkat tinggi. Pada umumnya Bahasa tingkat menengah menggunakkan instruksi-instruksi yang berupa symbol dalam memberikan perintah kepada komputer. Bahasa tingkat tinggi, merupakan Bahasa pemrograman yang memiliki instruksi-instruksi yang berasal dari Bahasa manusia, sehingga merupakan Bahasa pemrograman yang paling dekat dengan Bahasa manusia. Salah satu Bahasa pemrograman yang akan kami bahas berikut ini adalah Bahasa pemrograman tingkat tinggi, yakni Bahasa pemrograman Python.

Python adalah bahasa pemograman yang mampu berjalan di berbagai sistem operasi dan di berbagai arsitektur komputer. Bahasa pemrograman Python dibuat dan dikembangkan lebih lanjut oleh Guido Van Rossum, yaitu seorang programmer yang berasal dari Negara Belanda. Dibuat dan Dikembangkan pada tahun 1990 di Belanda lebih tepatnya di kota Amsterdam. Di tahun 1995 python kembali dikembangkan agar lebih kompatibel oleh Guido Van Rossum dan di awal tahun 2000 Versi Python kembali dikembangkan dan diperbaharui sehingga kini Bahasa Pemrograman Python telah mencapai Versi 3. Sebenarnya awal mula dari kata ‘Python‘ diambil oleh Guido Van Rossum dari sebuah acara televisi yang lumayan terkenal yang bernama Mothy Python Flying Circus, Yaitu sebuah acara sirkus yang disukai oleh Guido Van Rossum. Bahasa pemrograman Python juga merupakan sebuah bahasa pemrograman komputer yang mempunyai banyak kegunaan. Bahasa pemograman ini termasuk salah satu bahasa pemograman tingkat tinggi dan *object oriented* atau dikenal dengan berorientasi pada objek seperti Bahasa pemrograman C++ dan java. Python dapat digunakan untuk berbagai keperluan pengembangan perangkat lunak dan dapat berjalan di berbagai platform sistem operasi. Seperti halnya bahasa pemrograman dinamis, Python seringkali digunakan sebagai bahasa skrip dengan interpreter yang teintergrasi dalam sistem operasi. Saat ini kode Python dapat dijalankan pada sistem berbasis Linux/Unix, Windows, Mac OS X, Java Virtual Machine, Amiga, Palm, Symbian. Bahasa pemrograman python, secara umum telah jauh lebih baik dibandingkan dengan Bahasa pemrograman lainnya, hal ini dikarenakan sintaksis kode dan script nya yang mudah dipahami. Bahasa pemrograman Python merupakan gabungan/kombinasi dari beberapa Bahasa pemrograman lainnya, seperti java, C, C++, Bash, dan lainnya. Bahasa pemrograman Python juga mempunyai sebuah pelengkap bahasa pemrogramannya, yaitu pustaka standar yang fungsional dan juga sangat besar, dan juga komprehensif.

1. **Pengertian Syntax dan Semantik**
2. **Sintaks**

Sintaks sebuah bahasa berhubungan dengan struktur bahasa. Sebagai contoh, untuk membentuk sebuah kalimat yang valid dalam bahasa kita memakai struktur: [subyek] + [kata kerja] + [kata benda]. Dengan memakai struktur ini, kita bisa membentuk kalimat, sebagai contoh: Saya makan nasi. Dalam hubungannya dengan bahasa pemrograman, kita musti memenuhi sintaks (baca: aturan struktur bahasa) agar program dapat berjalan. Sebagai contoh, dalam bahasa BASIC, untuk mengassign sebuah variabel dengan sebuah nilai, kita memakai operand ‘=’, tetapi kalau dalam Pascal, kita pakai ‘:=’. Contoh dalam BASIC: a=1, tapi dalam bahasa Pascal, a:=1.

1. **Semantik**

 Semantik sebuah bahasa menggambarkan hubungan antara sintaks dan model komputasi. Sederhananya, semantik menjelaskan arti dari program.Analoginya sebagai berikut. Apabila kita memakai sintaks [subyek] + [kata kerja] + [kata benda], kita bisa menghasilkan kalimat-kalimat. Apabila kita mengasilkan kalimat Saya makan nasi, maka kalimat ini memenuhi aturan sintaks. Tapi, apabila saya membuat kalimat Saya makan batu, secara sintaks kalimat ini sudah benar. Namun, secara semantik, kalimat ini tidak mengandung makna yang berarti. Dalam hubungannya dengan bahasa pemrograman, kadang ada kalanya seorang programmer tidak bisa mengaitkan sintaks dengan model komputasi. Kesalahan logika bisa dengan mudah terjadi.

**BAB 1**

**Describing Syntax and Semantics**

* 1. **Comments**
	2. **Comment untuk satu baris**

Dilakukan dengan menggunakan tanda # pada awal comment

Contoh :

#ini adalah comment untuk satu baris

* 1. **Comment untuk beberapa baris**

Dilakukan dengan menggunakan tanda “”” dan ditututp dengan tanda “”” bisa juga menggunakan kutip ‘’’ dan diakhiri ‘’’

Contoh :

“”” ini adalah

 Comment untuk beberapa baris “””

 ‘’’ ini juga comment

beberapa baris ‘’’

* 1. **Whitespace**

Ruang kosong dalam suatu design.Contohnya : space , tabs, linefeed, return, formfeed and vertical tab

print “this is a test”, \ “of contuniation”

Prints

This is a test of continuation

* 1. **Indentation**

Dalam pemrograman python, sistem indentasi merupakan 1 atau lebih spasi yang digunakan untuk untuk memasukkan sub sub program ke dalam suatu blok. Indentasi dalam pemrograman python dapat dilakukan dengan menggunakan spasi saja ataupun menggunakan tabulasi.

if a = b :

 print a, 'sama dengan', b #Menggunakan 2 spasi

else :

print a, 'tidak sama dengan', b #Menggunakan 2 spasi

if a = b :

print a, 'sama dengan', b #Menggunakan tabulasi

else :

print a, 'tidak sama dengan', b #Menggunakan tabulasi

* 1. **Tokens**
	2. **Keyword**

|  |  |  |  |
| --- | --- | --- | --- |
| [False](http://www.programiz.com/python-programming/keyword-list#true_false) | [class](http://www.programiz.com/python-programming/keyword-list#class) | [finally](http://www.programiz.com/python-programming/keyword-list#finally) | [is](http://www.programiz.com/python-programming/keyword-list#is) |
| [None](http://www.programiz.com/python-programming/keyword-list#none) | [continue](http://www.programiz.com/python-programming/keyword-list#break_continue) | [for](http://www.programiz.com/python-programming/keyword-list#for) | [try](http://www.programiz.com/python-programming/keyword-list#except_raise_try) |
| [True](http://www.programiz.com/python-programming/keyword-list#true_false) | [def](http://www.programiz.com/python-programming/keyword-list#def) | [from](http://www.programiz.com/python-programming/keyword-list#from_import) | [nonlocal](http://www.programiz.com/python-programming/keyword-list#nonlocal) |
| [and](http://www.programiz.com/python-programming/keyword-list#and_or_not) | [del](http://www.programiz.com/python-programming/keyword-list#del) | [global](http://www.programiz.com/python-programming/keyword-list#global) | [not](http://www.programiz.com/python-programming/keyword-list#and_or_not) |
| [as](http://www.programiz.com/python-programming/keyword-list#as) | [elif](http://www.programiz.com/python-programming/keyword-list#if_else_elif) | [if](http://www.programiz.com/python-programming/keyword-list#if_else_elif) | [or](http://www.programiz.com/python-programming/keyword-list#and_or_not) |
| [assert](http://www.programiz.com/python-programming/keyword-list#assert) | [else](http://www.programiz.com/python-programming/keyword-list#if_else_elif) | [import](http://www.programiz.com/python-programming/keyword-list#from_import) | [while](http://www.programiz.com/python-programming/keyword-list#while) |
| [break](http://www.programiz.com/python-programming/keyword-list#break_continue) | [except](http://www.programiz.com/python-programming/keyword-list#except_raise_try) | [in](http://www.programiz.com/python-programming/keyword-list#in) | [return](http://www.programiz.com/python-programming/keyword-list#return) |

Merupakan suatu kata kunci yang telah di sediakan oleh program yang memiliki arti khusus dan tidak boleh digunakan untuk penamaan lainnya.

* 1. **Operator**

Python memiliki beberapa jenis operator yaitu, operator aritmatika, operator boolean, operator logic, operator relational. Berikut beberapa contoh operator yang ada pada pemrograman python:

|  |  |
| --- | --- |
| + | Penjumlahan |
| - | Pengurangan |
| \* | Perkalian |
| / | Pembagian |
| \*\* | Pangkat |
| // | Pembagian bulat kebawah |
| % | Modulus |
| << | Geser kiri |
| >> | Geser kanan |
| & | Bit wise and  |
| | | Bit wise or  |
| ^ | Bit wise xor |
| ~ | Bit wise invert |
| < | Kurang dari  |
| > | Lebih dari |
| <= | Kurang dari atau sama dengan |
| >= | Lebih dari atau sama dengan |
| == | Sama dengan |
| != | Tidak sama dengan |
| not | Negasi  |
| or | Atau  |

* 1. **Punctuator**

String dalam karakter ASCI. String mencakup huruf, whitespace, angka dan memiliki tanda baca yaitu koma, kutip dan titik koma.

import string

# Display punctuation.

for c in string.punctuation:

print("[" + c + "]")

**Output**

[!] ["] [#] [$] [%] [&] ['] [ (] [)] [\*] [+] [,] [-] [.] [/] [:] [;] [<] [=] [>] [?] [@] [[] [\] []] [^] [\_] [`] [{] [|] [}] [~]

import string

# An input string.

name = "hey, my friend!"

for c in name:

 # See if the char is punctuation.

 if c in string.punctuation:

print("Punctuation: " + c)

* 1. **Delimiter**

Untuk memisahkan atau mengelompokkan identifier. Sebagai contoh delimiter “ ( “ yang digunakan bersamaan dengan “ ) “ digunakan untuk mengelompokkan integer dalam operasi aritmatika seperti pada contoh berikut:

>>> a = 2

>>> b = 3

>>> c = 4

>>> print (a + b) \* c

20

>>> print a \* (b + c)

14

>>> print a \* b + c

10

* 1. **Literal**

Literal string dapat di awali dan di tutup dengan tanda kutip ganda atau tunggal. Sebuah string literal dapat dikutip ganda pada saat string mengandung tanda kutip tunggal(misalnya ‘i don’t know’ pada code (print “i don’t know”) dan juga pada string yang dikutip tunggal dapat berisi tanda kutip ganda.misalnya string (‘I “love” you’ pada code print “i \”love\” you”).

* 1. **Print**

Digunakan untuk menampilkan sesuatu ke layer. Ada beberapa cara digunakan untuk menggunakan keyword print, yaitu:

#Cara Pertama :

print "First number is {} and second number is {}".format(first, second)

#Cara Kedua :

print "First number is {first} and number is {second}".format(first=first, second=second)

#Cara Ketiga :

print 'First number is', first, ' second number is', second

#Cara Keempat :

print 'First number %d and second number is %d' % (first, second)

#Cara Kelima :

print 'First number is ' + str(first) + 'second number is ' + str(second)

* 1. **Raw Input**

Digunakan untuk menerima masukkan dari user.

X = raw\_input()

X = raw\_input(‘what is your name?’)

* 1. **If, elif, else (conditional)**

Digunakan untuk mengeksekusi statement statement tertentu dan hanya akan dieksekusi jika dan hanya jika kondisinya bernilai benar.

#Contoh 1

if a>1 and b>10 :

 print a

elif a<1 and b<10 :

 print b

else :

print ‘tidak ada’

#Contoh 2

if (a>1 and b>10) :

 print a

elif (a<1 and b<10) :

 print b

else :

print ‘tidak ada’

* 1. **True, False, not (ekspresi Boolean)**

Biasanya digunakan untuk mengekspresikan nilai boolean yang memiliki 2 jenis nilai yaitu True dan False.

a = True

b = False

if a = True or b = False :

 print a

elif a = not True or b = not False :

 print b

else :

print ‘tidak ada’

* 1. **Expressions**

Kombinasi dari nilai, variable, dan operator. Sebuah nilai saja sudah dinyatakan sebagai ekspresi, dan begitu juga sebuah variable, dibawah ini adalah ekspresi yang benar (jika diasumsikan variable x telah diberikan sebuah nilai).

17

x

x + 17

#Jika ekspresi diketik dalam mode interaktif phyton, ekspresi dapat di evaluasi dan memunculkan hasil:

>>> 1 + 1

2

#Tetapi dalam skrip, ekspresi sendiri tidak menghasilkan apa apa.

5

x = 5

y = x + 1

#tidak akan melakukan apa apa jika tidak diberi statement print

print (y)

* 1. **Intepreter Errors**

Berikut ini adalah beberapa kegunaan dari Intepreter Errors, yaitu:

* Mengubah kode python menjadi bahasa mesin, dan menentukan apakah kode tersebut valid atau tidak
* Jika kode tersebut tidak tepat. Python tidak dapat menjalankan kode tersebut dan mengeluarkan interpreter error.

>>>print (“tidak ada error di sini”)

#tidak ada error di sini

>>>print (ada error di syntax ini)

SyntaxError: invalid syntax

>>>print “ada error juga disini”
SyntaxError: invalid syntax

**BAB 2**

**NAME, BINDING, AND SCOPE**

1. **Nama Variabel**

 Pada umumnya, programmer memakai nama variabel sesuai dengan
keterangan isi dari variabel tersebut dan variabel juga merupakan simbol yang mewakili nilai tertentu. Pembuatan variabel dalam python sangat sederhana. Berikut adalah ketentuan mengenai variabel dalam pyton:

1. Variabel tidak perlu dideklarasikan mempunyai tipe data tertentu
2. Jenis data dalam variabel dapat berubah-ubah
3. Penulisan variabel harus diawali dengan huruf, dan untuk karakter selanjutnya bisa berupa huruf atau angka
4. Dapat berupa huruf Kapital, tetapi bersifat case-sensitive, nama Kapital dengan kapital adalah variabel yang berlainan.
5. Penulisan variabel tidak boleh dipisah oleh spasi
6. Untuk variabel yang terdiri dari 2 suku kata, dapat dipisah dengan simbol underscore ( \_ ) seperti nama\_saya, nama\_variabel\_nama.

Statemen yang tidak boleh dijadikan nama variabel adalah keywords pada Python. Contoh :

>>> 123satu = "angka"

variabel 123satu adalah penamaan variabel tidak benar karena diawali dengan sebuah angka.

>>> lebih$ = 50000

lebih$ juga tidak benar karena terdapat karakter yang tidak semestinya ada dalam penamaan variabel.

1. **Tipe dan Jenis Data**

 Tipe data adalah jenis data yang tersimpan dalam variabel. Python secara otomatis mengenali tipe data yang tersimpan dalam variabel. Contoh tipe data sebagai berikut:

* 1. **Tipe Data Bilangan**

 Di Python bilangan dibagi menjadi dua tipe utama yaitu integer (bilangan bulat) dan float (bilangan pecahan). Contoh tipe data integer: 28, 90, -43, 23 dsb., sedangkan contoh tipe data float (pecahan): 3.23, -1,213, 52.3e-4 dan sebagainya.

* 1. **Tipe Data Teks**

 Teks (String) merupakan jenis data yang terdiri dari sekumpulan karakter. Penulisan tipe data teks harus diapit dengan tanda petik: tunggal ('...'), ganda ("..."), dan rangkap tiga ('''...''' atau """...""").

* 1. **Tipe Data Boolean**

Tipe data boolean adalah tipe data yang hanya memiliki dua nilai yaitu True dan False atau 0 dan 1. Penulisan true dan false, huruf pertamnya harus kapital. Penulisan tipe data boolean sama seperti tipe data angka, tanpa menggunakan tanda petik.

1. **Binding**

Binding dalah asosiasi, antara atribut dan entitas atau antara operasi dan simbol. Binding statis jika terjadi sebelum run time dan tidak berubah selama eksekusi program. Jika terjadi saat run time dan bisa berubah selama eksekusi program, maka binding dinamis.

* 1. **Binding Type**

Berikut merupakan beberapa jenis Binding:

- Eksplisit, adalah pernyataan dalam program yang berisi daftar nama-nama variabel dan menentukan bahwa mereka adalah jenis tertentu.
- Implisit, assosiasi antara variabel dengan tipenya melalui cara biasa selain deklarasi.

* 1. **Storage Binding and Allocation**

- Static-Dynamic Variable, adalah variabel yang tempatnya dibuat saat statement deklarasinya dipanggil.

- Explicit heap-dynamic variable, memori tidak bernama yang dialokasi dan dealokasi oleh perintah eksplisit saat run time yang di berikan programmer.

- Implicit heap-dynamic variable, terikat pada memori jika diberi nama.

1. **Variable Scope**

 Pada Python Variable Scope adalah sebuah kondisi dimana variabel diakses secara lokal pada blok kode tertentu atau bersifat universal yang menyebabkan variabel tersebut dapat diakses dari blok kode manapun hasil.



Outputnya:



BAB 3

**DATA TYPES**

**Primitive Data Type**

Primitive data type adalah tipe data yang berbentuk primitive, dan pasti ada dalam setiap bahasa pemrograman. Contoh dari primitive data type :

- Integer

- Floating Point

- Complex

- Decimal

- Boolean

- Character

**Character String Type**

Character String Type adalah satu tipe di mana nilai-nilai terdiri dari urutan karakter. Karakter string yang konstan digunakan untuk label output, dan input dan output dari semua jenis data yang sering dilakukan dalam hal string. Adapun Premasalahan pada character string type :

o Apakah tipe primitif atau hanya jenis khusus array?

o Haruskah panjang string statis atau dinamis?

Tipe – tipe operasi :

o Assignment and copying

o Comparison (=, >, etc.)

o Catenation

o Substring reference

o Pattern matchin

**User Defined Ordinal Type**

 User defined ordinal type Adalah kisaran nilai yang mungkin dapat dengan mudah terkait dengan himpunan bilangan bulat positif. 3 tipe data user defined primitive :

o Integer

o Char

o Boolean

**Array Type**

array adalah agregat homogen elemen data di mana seorang individu elemen diidentifikasi oleh posisinya dalam agregat, relatif terhadap elemen pertama.

Subscript Binding and Array Categories:

- Static = rentang subscript yang statis terikat dan alokasi penyimpanan statis (dilakukan sebelum run time).

- Fixed stack-dynamic = rentang subscript yang statis terikat, namun alokasi dilakukan pada saat deklarasi elaborasi selama eksekusi.

- Stack-dynamic = rentang subscript dan alokasi penyimpanan dinamis terikat pada waktu elaborasi.

- Fixed heap-dynamic = rentang subscript dan storage binding keduanya tetap, setelah penyimpanan dialokasikan.

- Heap-dynamic = pengikatan rentang subscript dan alokasi penyimpanan dinamis dan dapat berubah beberapa kali selama seumur hidup array.

* 1. **Heterogeneous Arrays**

Heterogeneous array adalah satu di mana unsur-unsur tidak perlu dari jenis yang sama. Rectangular array adalah array multidimensional di mana semua baris memiliki jumlah yang sama elemen dan semua kolom memiliki jumlah yang sama dari elemen. Array Rectangular bermodelkan meja persegi panjang. jagged array adalah di mana panjang dari baris tidak perlu menjadi sama. Sebagai contoh, sebuah matriks jagged dapat terdiri dari tiga baris, satu dengan 5 elemen, satu dengan 7 elemen, dan satu dengan 12 elemen.

* 1. **Associative Arrays**

Associative Arrays adalah koleksi tak berurut elemen data yangdiindeks oleh jumlah yang sama nilai-nilai yang disebut kunci.

**Record Types**

record adalah jumlah elemen data di mana unsur-unsur individu diidentifikasi oleh nama dan diakses melalui offset dari awal struktur. Record adalah agregat mungkin heterogen elemen data di mana unsur-unsur individu diidentifikasi dengan nama

* + - 1. **Tuple Types**

Tuple adalah adalah tipe data yang mirip dengan catatan, kecuali bahwa unsur-unsur yang tidak bernama . Digunakan dalam Python, ML, dan F # untuk memungkinkan fungsi untuk kembali beberapa nilai

* + - 1. **List Type**

Lists pertama kali didukung dalam bahasa pemrograman fungsional LISP. Mereka selalu menjadi bagian dari bahasa-bahasa fungsional, tetapi dalam beberapa tahun terakhir mereka telah menemukan cara mereka ke dalam beberapa bahasa imperatif. Lists dalam Skema dan LISP dipisahkan oleh tanda kurung dan unsur tidak dipisahkan oleh tanda baca apapun. Sebagai contoh, (A B C D) Nested lists memiliki bentuk yang sama, sehingga kita bisa memiliki (A (B C) D).

* + - 1. **Union Type**

Union adalah jenis variabel yang dapat menyimpan nilai jenis yang berbeda pada waktu yang berbeda selama eksekusi program. Sebagai contoh kebutuhan untuk jenis serikat, mempertimbangkan tabel konstanta untuk compiler, yang digunakan untuk menyimpan konstanta ditemukan dalam program yang dikompilasi.

* + - 1. **Pointer and Reference Types**

Sebuah variabel tipe pointer memiliki berbagai nilai-nilai yang terdiri dari alamat memori dan nilai khusus, nil. Memliki kemampuan untuk indirect addressing . memberikan cara untuk me manage dynamic memory. Pointer dapat digunakan sebagai akses lokasi di area penyimpanan dinamis(Heap).

* + - 1. **Type Checking**

Type checking adalah kegiatan memastikan bahwa operan dari operator adalah dari jenis yang compatible . Compatible type adalah salah satu yang legal bagi operator, atau diperbolehkan di bawah aturan bahasa yang akan dikonversi secara implisit, dengan kode compiler yang dihasilkan, untuk tipe legal. Konversi otomatis ini disebut coercion. Type error adalah aplikasi operator untuk operan dari tipe yang tidak layak.

* + - 1. **Strong Typing**

Strong Typing adalah jika jenis kesalahan selalu terdeteksi. Ini mensyaratkan bahwa semua jenis operan dapat ditentukan, baik pada waktu kompilasi atau run time. Pentingnya String typing terletak pada kemampuannya untuk mendeteksi semua penyalahgunaan variabel yang menghasilkan jenis kesalahan. Strong typing juga memungkinkan deteksi, pada waktu berjalan, kegunaan dari jenis yang salah nilai dalam variabel yang dapat menyimpan nilai lebih dari satu jenis.

**BAB 4**

**EXPRESSION AND ASSIGNMENT STATEMENT**

**Operator Aritmatika**

Operator aritmatika pada python dapat mengolah data integer, string, float hingga boolean. Beberapa diantaranya seperti penjumlahan yang dilambangkan dengan +, pengurangan -, perkalian \*, pembagian / dan modulus %, perpangkatan \*\*.Integer meliputi bilangan positif dan negatif yang tidak memiliki nilai desimal.String tipe data yang digunakan untuk menyimpan barisan karakter Float merupakan tipe data yang digunakan untuk menyimpan bilangan pecahan. Boolean suatu tipe data yang hanya mempunyai dua nilai. Yaitu true atau false (benar atau salah).

2 + 3

2 - 3

2 \* 3

4 / 2

2 \*\* 3 #artinya 2 pangkat 3 hasilnya 8

10 % 3 # 10 mod 3 artinya sisa hasil bagi, akan menghasilkan 1

Jika salah satu dari dua bilangan yang dioperasikan berbentuk float, maka hasilnya akan float juga

10 / 3 #hasilnya integer 3

10.0 / 3 #hasilnya float 3.33333

10.0 / 2 #hasilnya float 5.0

Pada tipe data boolean True dan False dapat menghasilkan integer jika dijumlahkan, sebab True bernilai 1 dan False bernilai 0.

True + True #hasilnya 2

True + 2 #hasilnya 3

String pun dapat digunakan dalam operasi aritmatika, namun terbatas pada syarat tertentu, yaitu:

* String hanya bisa dijumlah dengan string
* String hanya bisa dikali dengan integer

"hai" + "Yo" #hasilnya 'haiYo'

"hai" \* 3 #hasilnya 'haihaihai'

#akan error jika:

"hai" / "yo"

Selain operator aritmatika, juga ada operator logika, contoh:

True and True #hasilnya True

True and False #hasilnya false

True or True #hasilnya True

True or False #hasilnya False

#contoh dengan angka:

8 < 1 #hasilnya adalah False

8 < 1 or 9 > 5 #hasilnya True

**2. Overloading dan Type Conversion**

* 1. **List**

List sering disebut juga array pada bahasa pemrograman lain. List adalah jenis data campuran yang bisa memiliki komponen penyusun yang berbeda-beda. Sebuah list dapat dibuat dengan menggunakan tanda kurung siku [ ]. Anggota list didaftar dalam kurung siku tersebut dan masing-masing dipisahkan oleh tanda koma. Sifat-sifat list bisa didaftar seperti ini:

1. Komponen penyusunnya bisa diganti.
2. Komponen penyusunya dapat dibaca dan dimanipulasi secara langsung.
3. Komponen penyusunnya bisa ditambah.
4. Komponen penyusunnya dapat diambil dengan menunjukkan indeksnya atau dengan notasi slice.
5. Komponen penyusun sebuah list dapat juga berupa list yang lain

>>> a = ['lab', 'TI', 2010, 'J1']

>>> a ['lab', 'TI', 2010, 'J1']

>>> a[3]

'J1'

>>> a[2]

2010

>>> a[1:-1]

 ['TI', 2010]

>>> a[:3] + ['gunadarma', 'ATA', 2009, 2010]

['lab', 'TI', 2010, 'gunadarma', 'ATA', 2009, 2010]

>>> len (a)

4

Python dapat mengelompokan beberapa tipe data yang berbeda menjadi satu kelompok, yang kemudian dikenal sebagai List pemisah tanda koma ",".

>>> a = ["satu", 2, 3.0, "empat"]

>>> print a

['satu', 2, 3.0, 'empat']

Lists bisa dianalogikan sebagai array dan urutan pengaksesannya dimulai dari 0.

>>> a[0]

'satu'

>>> a[1]

2

>>> a[-2]

3.0

>>> a[3]

'empat'

Pengaksesan List pada urutan terakhir dengan nilai -1

|'satu' | 2 | 3.0 |'empat'|

| a[0] | a[1] | a[2] | a[3] |

|'satu' | 2 | 3.0 |'empat'|

| a[-4] | a[-3] | a[-2] | a[-1] |

List juga dapat dipisah - pisahkan dan dapat digabungkan, ditambahkan dan lainnya.

>>> a[0:2]

['satu', 2]

>>> a[-4:-1]

['satu', 2, 3.0]

Tanda titik dua ":" mempunyai argumen [:], berarti dimulai dari indeks sampai indeks ke -n (batas indeks-n, tidak ditampilkan). Di tambahkan, Misalnya :

>>> a + ['lima', 'enam']

['satu', 2, 3.0, 'empat', 'lima', 'enam']

Penambahannya hanya dapat dilakukan antar lists. Begitupun operasi penggandaan suatu lists, sebagian anggota list ataupun salah satu anggota list.

>>> 3\*a[:3] + ['tujuh']

['satu', 2, 3.0, 'satu', 2, 3.0, 'satu', 2, 3.0, 'tujuh']

>>> [a[3]] + [a[2]] + ['delapan']

['empat', 3.0, 'delapan']

>>> print a[3] 'empat'

Untuk melakukan perubahan terhadap satu anggota atau sebagian anggota list , kita hanya mengassignkan nilainya, Misalnya :

>>> a[2]

3.0

>>> a[2] = a[1] + 5

>>> a

['satu', 2, 7, 'empat']

yang berarti nilai a[2] digantikan menjadi nilai a[1] = 2 ditambahkan dengan 5, maka nilai a[2] menjadi 7. Untuk menggantikan sebagian anggota list secara berurutan juga diperbolehkan. Misalnya :

>>> a[0:2] = [1,'dua']

#Menggantikan elemen a[0],

a[1] #Menjadi a[0] = 1, a[1] = 'dua'

>>> print a [1, 'dua', 7, 'empat']

#Menghilangkan beberapa elemen anggota.

>>> a[0:2] = []

>>> print a

[7, 'empat']

#Menyisipkan suatu nilai.

>>> a[0:-1] = ['satu']

>>> a

['satu', 'empat']

Contoh diatas, berarti menempatkan elemen di antara 0,1 sampai -1. Untuk mengetahui jumlah elemen anggota List, digunakan fungsi built-in len yang berlaku juga untuk menghitung character suatu string.

>>> len(a)

2

Untuk menambahkan anggota elemen list digunakan metode append yang berlaku pada list. Misalnya :

>>> a.append('lima')

>>> a.append('enam')

>>> a ['satu', 'empat', 'lima', 'enam']

#List di dalam List.

>>> b = ['tujuh']

>>> a.append(b)

>>> a ['satu', 'empat', 'lima', 'enam', ['tujuh']]

Berikut metode - metode yang dapat dilakukan dengan object List :

1. append (x) : Menambahkan satu elemen anggota dan diletakkan di bagian indeks akhir pada segment
2. extend (L) : Menggantikan seluruh anggota elemen pada List menjadi seluruh elemen list L
3. insert(i, x) : Menyisipkan satu elemen anggota List pada posisi tertentu
4. remove(x) : Menghilangkan satu anggota list
5. pop([i]) : Menghilangkan salah satu anggota tertentu yang telah ditentukan posisinya
6. index(x) : Mengembalikan nilai indeks suatu anggota list
7. count(x) : Memeriksa jumlah x di dalam List
8. sort() : Mensorting list atau mengurutkan anggota list
9. reverse() : Kebalikan dari fungsi sort()
10. **Relational Expression and Boolean**

Relational Expression membandingkan sebuah nilai dan menyatakan relasi antara kedua nilai tersebut.

|  |  |
| --- | --- |
| == | If the values of two operands are equal, then the condition becomes true. |
| != | If values of two operands are not equal, then condition becomes true. |
| <> | If values of two operands are not equal, then condition becomes true |
| > | If the value of left operand is greater than the value of right operand, then condition becomes true. |
| < | If the value of left operand is less than the value of right operand, then condition becomes true. |
| >= | If the value of left operand is greater than or equal to the value of right operand, then condition becomes true. |
| <= | If the value of left operand is less than or equal to the value of right operand, then condition becomes true. |

Boolean Expression adalah suatu ekspresi dalam bahasa pemrograman yang menghasilkan nilai Boolean (true atau false).

Boolean operator:

|  |  |  |  |
| --- | --- | --- | --- |
| Not |

|  |  |
| --- | --- |
|  | Used to reverse the logical state of its operand. |

 |
| And | If both the operands are true then condition becomes true. |
| Or | If any of the two operands are non-zero then condition becomes true. |

#Contoh relational exp

#!/usr/bin/python

a = 21

b = 10

c = 0

if ( a == b ):

 print "Line 1 - a is equal to b"

else:

 print "Line 1 - a is not equal to b"

if ( a != b ):

 print "Line 2 - a is not equal to b"

else:

 print "Line 2 - a is equal to b"

if ( a <> b ):

 print "Line 3 - a is not equal to b"

else:

 print "Line 3 - a is equal to b"

if ( a < b ):

 print "Line 4 - a is less than b"

else:

 print "Line 4 - a is not less than b"

if ( a > b ):

 print "Line 5 - a is greater than b"

else:

 print "Line 5 - a is not greater than b"

a = 5;

b = 20;

if ( a <= b ):

 print "Line 6 - a is either less than or equal to b"

else:

 print "Line 6 - a is neither less than nor equal to b"

if ( b >= a ):

 print "Line 7 - b is either greater than or equal to b"

else:

 print "Line 7 - b is neither greater than nor equal to b"

Output :

Line 1 - a is not equal to b

Line 2 - a is not equal to b

Line 3 - a is not equal to b

Line 4 - a is not less than b

Line 5 - a is greater than b

Line 6 - a is either less than or equal to b

Line 7 - b is either greater than or equal to b

## Short circuit evaluation of logical expressions

Ketika phyton memproses ekspresi seperti “x >= 2 and (x/y) > 2”, ekspresi dievaluasi dari kiri-ke-kanan. Karena definisi dari “and”, jika x kurang dari 2, ekspresi x >= 2 salah dan keseluruhan dari ekspresi bernilai salah tanpa memperdulikan (x/y) > 2 bernilai salah atau benar.

Ketika phyton mendeteksi tidak ada nilai yang perlu didapatkan dengan mengevaluasi ekspresi logika selanjutnya, evaluasi berhenti dan tidak mengevaluasi ekspresi logika sampai habis. Ketika evaluasi berhenti karena nilai keseluruhan telah diketahui, itu disebut **short-circuiting**.

>>> x = 6

>>> y = 2

>>> x >= 2 and (x/y) > 2

True

>>> x = 1

>>> y = 0

>>> x >= 2 and (x/y) > 2

False

>>> x = 6

>>> y = 0

>>> x >= 2 and (x/y) > 2

Traceback (most recent call last):

 File "<stdin>", line 1, in <module>

ZeroDivisionError: integer division or modulo by zero

>>>

Kalkulasi ke-3 gagal karena phyton mengkalkulasi “(x/y) and y” bernilai 0 yang menyebabkan runtime error. Tetapi contoh kedua tidak gagal karena bagian pertama ekspresi x >= 2bernilai salah sehingga (x/y) tidak pernah dieksekusi karena aturan **short circuit** dan tidak ada error.

1. **Assignment Statement**

Sebuah assignment statement menciptakan sebuah nama variable dan memberikannya sebuah nilai. Berikut adalah beberapa contoh Assignment Statement :

* pesan = "Hari ini pertama saya belajar bahasa pemprogram Python"
* n = 17
* pi = 3.1415926535897932

Contoh tersebut merupakan assignment statement, di mana string dengan isi  *"Hari ini pertama saya belajar bahasa pemprogram Python"* dimasukkan pada variable dengan nama *pesan,*  yang kedua memberi integer 17 pada n dan yang ketiga menetapkan (perkiraan) nilai π ke pi.

**BAB 5**

**CONTROL STRUCTURE AND STATEMENT**

1. **Statement-Level Control Structures**
	1. **Control Structure**

Adalah struktur yang digunakan untuk mengontrol output sesuai dengan input yang diberikan.

A.While loop, jika kondisi yang ditentukan bernilai benar, maka jalankan proses tersebut terus menerus sampai kondisi bernilai salah.

B. For, proses dijalankan berapa kali sesuai dengan nilai yang ditentukan sampai dengan batas

* 1. **Selection Statements**

Mengatur alur program berjalan berdasarkan suatu kondisi bernilai benar atau salah.

A.If, jika kondisi yang ditentukan bernilai benar, maka proses dijalankan.

B.If-else, jika kondisi yang ditentukan bernilai salah, maka proses kedua yang berada dalam else yang dijalankan.Switch-case, berbeda dengan if atau if else, switch case tidak dapat menilai kondisi, tetapi hanya membandingkan apa nilai sesuai.

* 1. **Two way selection statements**

If-else, hanya memiliki 2 kemungkinan, jika if benar, jalankan proses pertama, jika salah jalankan proses kedua yang terletak dalam else.

1. **Multiple way selection statements using switch case**

Kemampuan untuk mengubah perilaku sepotong kode yang didasarkan pada informasi tertentu di lingkungan dikenal sebagai conditional code flow. Conditional logic dalam Python terutama didasarkan pada struktur "if ... else".

* 1. **If statements**

If digunakan untuk menguji kondisi, jika kondisi benar maka kode dalam if dijalankan, sebaliknya jika salah, maka kode yang ada dalam if tidak dieksekusi.

if <condition> :

<statement>



i=1

if i==1:

print  "Hello intellipaat"

**Output**

Hello intellipaat

* 1. **If Else**

If else juga digunakan dalam menguji condition, jika condition benar maka kode dalam if dijalankan, sebaliknya jika salah, maka kode dalam else yang dijalankan

If <condition>:

<statements>

else:

<statements>



i=1

if i==2:

print  "Hello intellipaat"

else:

print “Hello”

**Output**

Hello

* 1. **Elif**

elif mengandung banyak if else condition. Ini digunakan untuk melakukan pengecekan pada kondisi yang terdapat lebih dari satu. Hal ini hamper sama dengan menjalankan statement if didalam statement else.

if <condition1>:

<statement>

if <condition2>:

<statement>

elif <condition3>:

<statement>

else

<statement>

else

<statement>

i = 1

if i>5:

print “value is more than 5”

elif i==5:

print “value is 5”

else

print “value is less than 5”

**Output**

value is less than 5

* 1. **Python Break**

Merupakan statement yang digunakan untuk mengakhiri secara paksa sebuah loop. Sehingga program akan langsung menjalankan kode setelah loop.



for i in [10,20,30,40]:

if i==30:

print "Item Found"

break

print i

**Output**

10

20

Item Found

* 1. **Python Continue**

Statement ini hamper sama dengan break yaitu untuk mengakhiri secara paksa sebuah loopingan. Akan tetapi, yang membedakannya ialah setelah program menjalankan statement continue, program tidak akan langsung menjalankan kode setelah loop, tetapi program akan kembali ke head of loop lagi (dapat dikatakan kode dalam looping setelah loop tidak dijalankan/di ‘skip’) dan kemudian menjalankan perulangan selanjutnya.



for i in [10,20,30,40]:

if i==30:

print "Item Found"

continue

print i

**Output**

10

20

Item Found

40

1. **Clause Form**
	1. **If Statement**

Kalau program dijalankan dan tidak ketemu true, maka hasil yang akan keluar berasal dari else.

**if\_stmt** ::= "if" [**expression**](https://docs.python.org/2/reference/expressions.html#grammar-token-expression) ":" [**suite**](https://docs.python.org/2/reference/compound_stmts.html#grammar-token-suite)

 ( "elif" [**expression**](https://docs.python.org/2/reference/expressions.html#grammar-token-expression) ":" [**suite**](https://docs.python.org/2/reference/compound_stmts.html#grammar-token-suite) )\*

 ["else" ":" [**suite**](https://docs.python.org/2/reference/compound_stmts.html#grammar-token-suite)]

* 1. **While Statement**

 Berulang kali menguji ekspresi dan, jika true, mengeksekusi suite pertama; jika ekspresi adalah false (yang mungkin pertama kali diuji) dari clauses suite yang lain, jika ada, dijalankan dan loop berakhir.

 Sebuah pernyataan istirahat dieksekusi di suite pertama berakhir loop tanpa mengeksekusi suite lain klausul ini. Sebuah melanjutkan pernyataan dieksekusi di suite pertama melompati sisa suite dan kembali menguji ekspresi.

**while\_stmt** ::= "while" [**expression**](https://docs.python.org/2/reference/expressions.html#grammar-token-expression) ":" [**suite**](https://docs.python.org/2/reference/compound_stmts.html#grammar-token-suite)

 ["else" ":" [**suite**](https://docs.python.org/2/reference/compound_stmts.html#grammar-token-suite)]

* 1. **For Statement**

 Untuk pernyataan digunakan untuk iterate atas unsur-unsur urutan (seperti string, tupel atau daftar) atau objek iterable lainnya:

 Daftar ekspresi dievaluasi sekali; itu harus menghasilkan sebuah objek iterable. Iterator diciptakan untuk hasil expression\_list tersebut. Suite ini kemudian dieksekusi sekali untuk setiap item yang disediakan oleh iterator, di urutan indeks naik. Setiap item pada gilirannya ditugaskan untuk daftar target menggunakan aturan standar untuk tugas, dan kemudian suite dijalankan. Ketika item habis (yang segera ketika urutan kosong), suite di lain klausul, jika ada, dijalankan, dan loop berakhir.

 Sebuah pernyataan istirahat dieksekusi di suite pertama berakhir loop tanpa mengeksekusi suite lain klausul ini. Sebuah melanjutkan pernyataan dieksekusi di suite pertama melompati sisa suite dan berlanjut dengan item berikutnya, atau dengan yang lain klausul jika tidak ada item berikutnya. Suite dapat menetapkan ke variabel (s) dalam daftar target; ini tidak mempengaruhi item berikutnya yang ditugaskan untuk itu.

 Daftar target tidak dihapus ketika loop selesai, tetapi jika urutan kosong, tidak akan ditugaskan sama sekali oleh loop. Petunjuk: built-in berbagai fungsi () mengembalikan urutan bilangan bulat cocok untuk meniru efek dari Pascal untuk i: = a ke b dilakukan; misalnya, rentang (3) mengembalikan daftar [0, 1, 2] ..

**for\_stmt** ::= "for" [**target\_list**](https://docs.python.org/2/reference/simple_stmts.html#grammar-token-target_list) "in" [**expression\_list**](https://docs.python.org/2/reference/expressions.html#grammar-token-expression_list) ":" [**suite**](https://docs.python.org/2/reference/compound_stmts.html#grammar-token-suite)

 ["else" ":" [**suite**](https://docs.python.org/2/reference/compound_stmts.html#grammar-token-suite)]

1. **Try Statement**

 Seperti if else Misalkan dalam kode berikut terjadi pembagian yang membagi suatu angka dengan nol. Sudah menjadi ketentuan bahwa jika sebuah angka dibagi nol maka program akan *error*. Oleh karena itu kita kurung dengan **try..except**, kemudian kita keluarkan *error*-nya begitu *error* tertangkap oleh **except**. Baris "print x+1" akan tetap dieksekusi hanya saja akan muncul *error* yang ditimbulkan oleh pembagian nol.

x = 0

**try**:

 x = 1 / 0

except **Exception**, e:

**print** e

**print** x + 1

Jika kode diatas dieksekusi, maka akan muncul *error* seperti berikut ini:

$ python demo-1.py

integer division **or** modulo **by** zero

1

* 1. **Menangani error pembacaan file**

 Di kode ini kita akan mencoba menangkap dua *error* pada kode yang dikurung oleh **try..except**. Terdapat sebuah *dictionary* yang berisi *key* **nama**, **kota**, dan **umur**. Kemudian kita membuka sebuah *file* yang bernama **contact.txt**. Walaupun ada kode *error* setelahnya yang akan mengakibatkan *error* pengaksesan indeks, yang akan ditangkap terlebih dahulu adalah *error*yang diakibatkan gagalnya membaca *file*.

orang = {"nama":"syuaib", "kota":"jepara", "umur":"20"}

**try**:

 contact = open("contact.txt", 'r')

**print** orang["pekerjaan"]

**except** IOError, e:

**print**"Terjadi error IO: ", e

**except** KeyError, e:

**print**"Terjadi kesalahan pada akses list/dict/tuple:", e

**print** orang

Bila kita eksekusi kode diatas, maka akan muncul *output* seperti berikut:

$ python demo-2.py

Terjadi error IO: [Errno 2] No such file **or**directory:'/home/contact.txt'

{'nama': 'syuaib', 'umur': '20', 'kota': 'jepara'}

 Megenal kata Finally

orang = {"nama":"syuaib", "kota":"jepara", "umur":"20"}

**try**:

**print** orang["nama"]

**except** KeyError, e:

**print**"Terjadi error KeyError: ", e

**finally**:

**print**"baris ini akan selalu dieksekusi"

**print** orang

Jika kode diatas kita eksekusi, maka akan muncul *output* seperti berikut:

$ python demo-3.py

syuaib

baris ini akan selalu dieksekusi

{'nama': 'syuaib', 'umur': '20', 'kota': 'jepara'}

Bila kita coba ganti "nama" dengan "pekerjaan", maka akan muncul *output*seperti berikut:

$ python demo-3.py

Terjadi error KeyError:'pekerjaan'

baris ini akan selalu dieksekusi

{'nama': 'syuaib', 'umur': '20', 'kota': 'jepara'}

 Mengenal kata raise

nilai = -999999

**try**:

**print**"baris ini akan di print"

**if** nilai <0:

**raise** ValueError, nilai

**print**"baris ini tidak akan di print"

**except** Exception, e:

**print** e

**finally**:

**print**"Ada error mas"

Bila kode diatas kita eksekusi di konsol maka akan muncul *output* seperti berikut:

$ python demo-4.py

baris ini akan di print

-999999

Ada error mas

1. **With Statement**

**with\_stmt** ::= "with" with\_item ("," with\_item)\* ":" [**suite**](https://docs.python.org/2/reference/compound_stmts.html#grammar-token-suite)

**with\_item** ::= [**expression**](https://docs.python.org/2/reference/expressions.html#grammar-token-expression) ["as" [**target**](https://docs.python.org/2/reference/simple_stmts.html#grammar-token-target)]

Pelaksanaan dengan pernyataan dengan satu "item" hasil sebagai berikut:

1.Ekspresi konteks (ekspresi diberikan dalam with\_item yang) dievaluasi untuk mendapatkan manajer konteks.

2.Konteks manajer \_\_exit \_\_ () dimuat untuk digunakan nanti.

3.\_\_enter \_\_ () metode konteks manajer dipanggil.

4.Jika target termasuk dalam dengan pernyataan, nilai kembali dari \_\_enter \_\_ () ditugaskan untuk itu.

Catatan: dengan pernyataan jaminan bahwa jika metode \_\_enter \_\_ () mengembalikan tanpa kesalahan, maka \_\_exit \_\_ () akan selalu disebut. Dengan demikian, jika terjadi kesalahan selama tugas untuk daftar target, ia akan diperlakukan sama dengan kesalahan yang terjadi dalam suite akan. Lihat langkah 6 di bawah ini.

5. Suite dijalankan.

6.\_\_exit \_\_ () metode konteks manajer dipanggil. Jika pengecualian disebabkan suite untuk keluar, jenis, nilai, dan traceback dilewatkan sebagai argumen untuk \_\_exit \_\_ (). Jika tidak, tiga ada argumen disediakan.

**with**A()**as**a,B()**as**b:

suite

sama dengan

**with**A()**as**a:

**with**B()**as**b:

suite

1. **Iterative statements**

from \_\_future\_\_ import print\_function

import random

oddCount= 0

for s in range(100):

 spinCount= s

 n= random.randrange(38)

 # Zero

 if n == 0 or n == 37: # treat 37 as 00

 oddCount = 0

 continue

 # Odd

 if n%2 == 1:

 oddCount += 1

 if oddCount == 6: break

 continue

 # Even

 assert n%2 == 0 and 0 < n <= 36

 oddCount = 0

print( oddCount, spinCount )

Jika kita memperlakukan 37 seolah-olah 00, yang seperti nol. Dalam Roulette, kedua angka tersebut tidak aneh. oddCount diatur ke nol, dan loop dilanjutkan. terus Pernyataan ini melanjutkan loop dengan nilai berikutnya yaitu s. pengulangan pengolahan terjadi di bagian atas untuk pernyataan/ **break** statement akan memberhentikan pernyataan di dalam pernyataan for.Sedangkan continue statement untuk mengatur kembali pernyataan for dengan value berikutnya yaitu s

1. **Counter Controlled loops**

for<target>in<object>:

<statement>

Dalam for ini terdapat 2 data, yaitu target yang nantinya berupa variable yang akan menampung setiap iterasi dari objek. Misal objek berisi list dari 1 sampai 100, maka target akan menampung nilai isi dari list satu per

Cth untuk memprint 1 – 100

for x in range(100):

print x+1# karena range dimulai dari 0 – 99

1. **Logically Controlled Loops**

Python mempunyai 3 jenis Loop yaitu

For Loop, While Loop dan Nested loop

* 1. **For Loop**

Seperti pada bahasa pemrograman lainnya, for loop sudah menjadi standar namun berbeda-beda tata cara penulisan nya di setiap pemrograman.

Contoh .

# Program mencetak angka 1 s/d 10

i = 10

for i in range(10):

 print(i+1)

 i = i+1

* 1. **While Loop**

While dipakai untuk looping dimana iterasi akan dilakukan selama kondisi yang diberikan benar. While ini juga bisa di pakai untuk Infinite loop.

Contoh:

# Program mencetak angka 1 s/d 10

i = 1

while(i < 11):

 print(i)

 i = i+1

1. **User Loaded Loop Control Mechanisms**
	1. **Sistem Kontrol Loop Terbuka**

 Sistem kontrol loop terbuka adalah suatu sistem yang keluarannya tidak mempunyai pengaruh terhadap aksi kontrol. Artinya, sistem kontrol terbuka keluarannya tidak dapat digunakan sebagai umpan balik dalam masukkan. Sistem kontrol terbuka dapat digunakan hanya jika hubungan antara masukan dan keluaran diketahui dan tidak terdapat gangguan internal maupun eksternal. Contoh Aplikasi Sistem Loop Terbuka : Pengontrol lalu lintas berbasis waktu, Mesin cuci, Oven listrik dan sebagainnya.

* 1. **Sistem Kontrol Loop Tertutup**

 Sistem Kontrol loop tertutup adalah sistem kontrol yang sinyal keluarannya mempunyai pengaruh langsung pada aksi pengontrolan. Atau loop tertutup berarti menggunakan aksi umpan balik untuk memperkecil kesalahan sistem. Contoh Aplikasi Sistem Kontrol Loop Tertutup : Servomekanisme, Sistem pengontrol proses, AC dan sebagainya.

**BAB 6**

**SUBPROGRAM**

**Prosedure dalam Subprogram**

Prosedure adalah bagian dari suatu program yang disusun secara terpisah untuk melakukan suatu tugas khusus/fungsi tertentu. Pada dasarnya ada dua macam bentuk prosedur yaitu Subrutin (subprogram) dan Fungsi.

Subrutin (Subprogram) adalah bagian dari program yang dibuat terpisah untuk melaksanakan sabagian dari tugas yang harus diselesaikan oleh suatu program. Pada umumnya yang dikenal dengan nama “prosedur” sebenarnya adalah ”subprogram”.

Fungsi adalah bagian dari program yang dibuat terpisah untuk melaksanakan fungsi tertentu yang menghasilkan suatu nilai untuk dikembalikan ke program utama.

Manfaat pembuatan prosedur :

1. Modularisasi

 Suatu program yang besar dan kompleks dapat dibagi kedalam beberapa prosedure sehingga setiap prosedure merupakan bagian yang mudah dikerjakan.

1. Simplifikasi

 Dalam suatu program, sering diperlukan suatu tugas yang harus dikerjakan berulang-ulang dengan nilai-nilai variabel yang berbeda-beda. Agar tidak merepotkan maka tugas ini cukup ditulis sekali saja dalam bentuk prosedur yang kemudian dipanggil berulang-ulang sesuai kebutuhan.

Penggunaan prosedur pada suatu program menyebabkan munculnya dua kategori variabel yaitu, Variabel Lokal yaitu variabel yang hanya dikenal dan berlaku dalam suatu  prosedur saja. Variabel Global yaitu variabel yang berlaku di semua bagian program dan di semua prosedur. Semua variabel yang didefinisikan pada deklarasi suatu prosedur adalah variabel lokal, dan variabel-variabel yang didefinisikan pada deklarasi algoritma utama adalah variabel global. Ketika suatu prosedur dipanggil maka pada hakikatnya bisa dilakukan pertukaran data antara program utama dan prosedur. Pertukaran ini dilakukan melalui parameter.

Parameter terbagi atas 2 macam, yaitu parameter actual adalah parameter yang disertakan pada saat prosedur dipangggil untuk dilaksanakan, sering disebut argumen. Parameter formal adalah parameter yang dituliskan pada definisi suatu prosedur/fungsi.

Ada 3 jenis parameter formal, yaitu parameter masukan (Input) adalah parameter yang menerima nilai dari parameter aktual. Parameter Keluaran (Output) adalah parameter yang menyerahkan nilai ke parameter actual. Parameter masukan dan keluaran (Input-Output) adalah parameter yang menerima nilai dari parameter aktual untuk diproses dalam prosedur kemudian diserahkan kembali ke parameter aktual setelah selesai.

**Fungsi dalam Subprogram**

Fungsi pada hakekatnya serupa dengan prosedur dalam pelaksanaannya, tetapi harus mengembalikan suatu nilai ke program yang menggunakan fungsi. Prosedur hanya bisa mengembalikan nilai melalui parameter input-output (in-out).

def function-name(Parameter list):

 statements, i.e. the function body

Parameter List bisa kosong atau terdiri dari banyak parameter. Parameter juga disebut argumen, jika fungsi ini dipanggil. Tubuh fungsi terdiri dari statement yang diindentasikan. Tubuh fungsi akan dijalankan setiap kali sebuah fungsi dipanggil. Parameter yang terdapat dalam fungsi bisa bersifat wajib atau opsional. Parameter opsional (nol atau lebih dari nol) harus mengikuti mandatory parameter.

Tubuh fungsi dapat berisi satu atau lebih return statement. Mereka dapat terletak di mana saja di tubuh fungsi. Return statement akan mengakhiri eksekusi tubuh fungsi dan kemudian me-return hasilnya, yaitu nilai dari ekspresi berikut keyword return, ke pemanggil fungsi. Jika return statement tanpa argumen, maka special value “None” yang akan di-return. Jika tidak ada return statement dalam kode fungsi, saat fungsi berakhir, ketika flow kontrol mencapai akhir tubuh fungsi dan nilai "None" akan di-return.

Example:

def fahrenheit(T\_in\_celsius):

 """ returns the temperature in degrees Fahrenheit """

 return (T\_in\_celsius \* 9 / 5) + 32

for t in (22.6, 25.8, 27.3, 29.8):

 print(t, ": ", fahrenheit(t))

Outputnya :

22.6 : 72.68

25.8 : 78.44

27.3 : 81.14

29.8 : 85.64

Fungsi dapat memiliki parameter opsional, atau disebut juga parameter default. Parameter default adalah sebuah parameter, yang tidak harus diberikan, jika sebuah fungsi dipanggil. Dalam hal ini, nilai-nilai default yang akan digunakan. Berikut ini merupakan prinsip operasi parameter default beserta dengan contohnya. Berikut script, yang tidak sangat berguna, menyapa seseorang. Jika tidak ada nama yang diberikan, fungsi akan menyapa “everybody”:

def Hello(name="everybody"):

 """ Greets a person """

 print("Hello " + name + "!")

Hello("Peter")

Hello()

Outputnya :

Hello Peter!

Hello everybody!

Menggunakan keyword parameter adalah cara alternatif untuk membuat panggilan fungsi. Definisi dari fungsinya tidak berubah.
An example:

def sumsub(a, b, c=0, d=0):

 return a - b + c - d

print(sumsub(12,4))

print(sumsub(42,15,d=10))

keyword parameter dapat digunakan dengan syarat, keyword tidak digunakan sebagai argumen posisional. Kita bisa melihat manfaat keyword parameter dalam contoh. Panggilan kedua untuk fungsi akan membutuhkan semua empat argumen, meskipun c perlu hanya nilai default:

print(sumsub(42,15,0,10))

Dalam contoh kami sebelumnya, kami menggunakan pernyataan kembali dalam fungsi sumsub tapi tidak di fungsi Hello. Jadi, dapat disimpulkan bahwa tidak wajib untuk memiliki return statement. Tapi apa yang akan di-return, jika kita tidak memberikan pernyataan kembali secara eksplisit. Contoh:

def no\_return(x,y):

 c = x + y

res = no\_return(4,5)

print(res)

Jika kita mulai script ini, maka yang akan dicetak adalah “None”, yaitu special value “None” yang akan di-return oleh fungsi return-less. “None” juga akan di-return, jika kita hanya memiliki return pada fungsi tanpa ekspresi:

def empty\_return(x,y):

 c = x + y

 return

res = empty\_return(4,5)

print(res)

Sebaliknya nilai dari ekspresi berikut dengan return yang akan di-return :

def return\_sum(x,y):

 c = x + y

 return c

res = return\_sum(4,5)

print(res)

**Design Issues of Subprograms**

* Lokal variable itu statis atau dinamis?
* Bisakah arti dalam sebuah subprogram muncul dalam subprogram lainnya?
* Apa metode parameter passing yang disediakan?
* Apakah tipe parameter di cek?
* Jika sebuah subprogram bisa dijadikan sebagai parameter dan subprograms bisa nested, apa referensi environmen sebuah subprogram?
* Bisakah subprogram overload?
* Bisakah subprogram menjadi generic?

**Local Referencing Environments**

* 1. **Local Variable**

Keuntungan dan kerugian penggunaan Stack Dynamic:

Advantage:

* Mendukung rekursi
* Data penyimpanan terbagi diantara beberapa subprogram

Disadvantage:

Alokasi / dealokasi, inisialisasi waktu

* + - Addressing yang tidak langsung
		- Subprogram tidak bisa sensitive

**4.2 Static Variable**

Advantage:

* Tidak ada runtime
* Lebih efisien

Disadvantage:

* Tidak bisa rekursi
	1. **Nested Subprograms**

def maker(n):

def action(x):

return x \*\* n

return action

f = maker(2)

print(f)

print(f(3))

print(f(4))

g = maker(3)

print(g(3))

print(f(3)) # still remembers 2

Semua variabel yang berasal dari fungsi orang tua digantikan oleh nilai aktual mereka di dalam fungsi anak. Dengan cara ini, tidak perlu untuk melacak lingkup fungsi orangtua untuk membuat fungsi anak berjalan dengan benar.

def maker(n):

def action(x):

return x \*\* n

return action

f = maker(2)

-->def action(x):

-->return x \*\*2

a =1

b =2

a, b = b, a

Python membacanya menjadi

a, b =2,1

Pada dasarnya memasukkan value sebelum dilakukan apapun.

**Parameter passing methods**

**5.1 Semantic model of parameter passing**

****

* 1. **Parameter pass by object**

 Pada python terdapat objek tetap maupun tidak tetap sehingga parameter passingnya didasarkan oleh objek. Didalam python value memiliki tempat tersendiri maupun jenis sendiri. Tidak ada penyalinan maupun referensi ke variable. Pemberian 2 nama pada python tidak masalah.

* 1. **type checking parameters**

 Karena tidak adanya tipe variable pada python maka type checking pada python juga tidak ada.

* 1. **example parameter passing in python**

def double\_it (formal\_parameter);

 print(formal\_parameter,id(formal\_parameter))

 doubled = formal\_parameter \* 2

 print (doubled,id(doubled))

 return doubled

 argument\_value = 3

 print(argument\_Value ,id(argument\_value))

 returned\_value = double\_it(argument\_value)

 print (returned\_value,id)returned\_value))

output

>>>

3 1410257392

3 1410257392

6 1410257740

6 1410257740

>>>

* 1. **Multidimensional array as parameters**

 Pada python tidak ada array. Namun sebagai gantinya python menyediakan struktur data list. Inisialisasi untuk list sama seperti variabel biasa. List dapat menampung tipe data berbeda dalam satu variabel.

Example 1, list as parameter:

def dosomething( thelist ):

for element in thelist:

print element

dosomething(['1','2','3'])

alist =['red','green','blue']

dosomething( alist )

output:

1

2

3

red

green

blue

Example 2, list as parameter:

def myfunc(a,b):

if(a>b):return a

else:return b

vecfunc = np.vectorize(myfunc)

result=vecfunc([[1,2,3],[5,6,9]],[7,4,5])

print(result)

output:

[[745]

[769]]

1. **Generic supprogram**

Penggunaan kembali perangkat lunak dapat menjadi kontributor penting untuk produktivitas software. Satucara untuk meningkatkan usabilitas dari perangkat lunak adalah untuk mengurangi kebutuhan untuk membuat yang berbedasubprogram yang menerapkan algoritma yang sama pada berbagai jenisdata. Sebagai contoh, seorang programmer tidak perlu menulis empat jenis yang berbedasubprogram untuk mengurutkan empat array yang berbeda hanya dalam jenis elemen. Sebuah subprogram polimorfik mengambil parameter dari jenis yang berbeda pada berbagai aktivasi. Subprogram kelebihan beban menyediakan jenis tertentu polimorfismedisebut ad hoc polymorphism.

Semacamyang lebih umumdaripolimorfismedisediakan olehmetodePythonsertaRuby. Variabel dalambahasa initidak memilikijenis,sehinggaparameter formaltidak memilikijenis. Oleh karena itu, metodeakan bekerja untuk setiapjenisparameteryang sebenarnya, asalkanoperatoryang digunakan padaparameter formaldalam metodedidefinisikan.

1. **Polymorphism(Overloading)**

 Polymorphism merupakan kemampuan suatu method untuk bekerja dengan lebih dari satu tipe argumen. Pada bahasa lain (khususnya C++), konsep ini sering disebut dengan method overloading. Pada dasarnya, Python tidak menangani hal ini secara khusus. Hal ini disebabkan karena Python merupakan suatu bahasa pemrograman yang bersifat dynamic typing yaitu tidak memerlukan deklarasi tipe. Polymorphism adalah suatu object dapat memiliki berbagai bentuk, sebagai object dari class sendiri atau object dari superclassnya.

**Overloading** : Penggunaan satu nama untuk beberapa method yang berbeda (beda parameter) **Overriding** : terjadi ketika deklarasi method subclass dengan nama dan parameter yang sama dengan method dari superclassnya.

Keuntungan Pemograman dengan menggunakan Polymorphism adalah :

* Kita dapat menggunakan kelas-kelas yang kita buat (sebagai super kelas) dan membuat kelas kelas baru berdasar superkelas tersebut dengan karakteristik yang lebih khusus dari behaviour umum yang dimiliki superkelas.
* Kita dapat membuat super kelas yang hanya mendefinisikan behaviuor namun tidak memberikan implementasidari metode-metode yang ada.Hal ini berguna jika kita ingin membuat semacam template kelas,kelas semacam ini disebut kelas abstrak karena behaviournya masih abstrak dan belum diimplementasikan.subkelas-subkelas dari kelas semacam ini yang disebut kelas konkret,mengimplementasikan behaviuor abstrak tersebut sesuai dengan kebutuhan masing-masing.
* Menghindari duplikasi object,kita dapat menciptakan class baru dari class yang sudah ada,sehingga tidak perlu menuliskan code dari nol ataupun mengulangnya, namun tetap bisa menambahkan attribute dan atau method unik dari class itu sendiri. Dalam konsep yang lebih umum sering kali polymorphism disebut dalam istilah satu interface banyak aksi.

Polymorphism dapat berarti banyak bentuk, maksudnya yaitu kita dapat menimpa (override), suatu method, yang berasal dari parent class (super class) dimana object tersebut diturunkan, sehingga memiliki kelakuan yang berbeda.

Pada dasarnya ada 2 tipe polymorphism, yaitu:

* Static atau trivial merupakan, function overloading (penggunaan kembali nama fungi yang sama tapi dgn argumen yang berbeda) yang terbagi dalam 3 signature yaitu: – Jenis Array
– Letak Array
– Type Array

 Contoh function overloading :

void tambah (int a, int b);
void tambah(float d, float c);

* Dynamic atau trueMerupakan function overriding (sebuah fungsi dalam class turunan yang memiliki nama, return type argumen function yang sama dengan fungsi dalam class induk). Menggunakan virtual method. - Pure Virtual Method (tanpa function body)
 contoh : virtual void jalan() = 0; - Squasi Virtual Method (ada function body)
 contoh : virtual void info() {;}

BAB 7

**ABSTRACT DATA TYPE**

1. **Introduction and Concept of Abstraction**

Abstraksi adalah pandangan atau representasi dari entitas yang hanya mencakup kebanyakan atribut signifikan. Dalam pengertian umum, abstraksi memungkinkan seseorang untuk mengumpulkan instansi dari entitas dalam kelompok-kelompok di mana atribut umum mereka tidak perlu menjadi dipertimbangkan. Misalnya, kita mendefinisikan burung sebagai makhluk dengan atributnya sebagai berikut ini: dua sayap, dua kaki, ekor, dan bulu. Kemudian, jika kita mengatakan gagak adalah burung, deskripsi gagak tidak perlu menyertakan atribut tersebut. Hal yang sama berlaku untuk burung robin, burung pipit, dan jenis burung lainnya. Atribut-atribut umum dalam deskripsi spesies tertentu burung dapat diabstraksikan, karena semua spesies burung memiliki semua atribut tersebut. Dalam spesies tertentu, hanya atribut yang dimiliki yang kemudian menjadi pembeda spesies satu dengan yang lainnya. Misalnya, gagak memiliki atribut berbulu hitam, memiliki ukuran tertentu, dan suka berkicau. Sebuah deskripsi dari gagak perlu untuk menyebutkan atribut yang dimiliki gagak secara spesifik.

Dalam dunia bahasa pemrograman, abstraksi adalah senjata melawan kompleksitas pemrograman; tujuannya adalah untuk mempermudah dalam proses pemrograman. Ini adalah senjata yang efektif karena memungkinkan programmer untuk fokus pada atribut penting, sementara mengabaikan atribut lainnya.

Terdapat dua jenis dasar abstraksi dalam bahasa pemrograman kontemporer yaitu, proses abstraksi dan abstraksi data. Konsep proses abstraksi adalah salah satu konsep tertua di design programming language. Semua subprogram merupakan proses abstraksi karena mereka menyediakan cara bagi program untuk menspesifikasikan sebuah proses, tanpa memberikan rincian tentang bagaimana subprogarm melakukan tugasnya. Misalnya, ketika sebuah program perlu memilah suatu data array numerik, biasanya digunakan subprogram untuk menyortir proses. Pada titik di mana proses penyortiran diperlukan, maka statement seperti misalkan *sortInt (daftar, listLen)* ditempatkan dalam program. Pemanggilan ini merupakan abstraksi dari proses penyortiran yang sebenarnya, yang dimana algoritma nya tidak dispesifikasikan. Panggilan ini juga bersifat independen terhadap algoritma diimplementasikan dalam pemanggilan subprogram. Dalam kasus subprogram *sortInt*, satu-satunya atribut penting ialah nama dari array yang akan diurutkan, tipe dari elemen elemennya, panjang arraynya, dan fakta bahwa panggilan untuk *sortInt* akan menghasilkan array yang telah diurutkan. Algoritma yang ada dalam subprogram *sortInt* tidak penting bagi user. User hanya perlu melihat nama dan protokol dari subprogram *sortInt* untuk dapat menggunakannya. Meluasnya penggunaan abstraksi data yang tentu diikuti dari proses abstraksi karena merupakan bagian penting dari setiap abstraksi data dalam operasinya, yang didefinisikan sebagai proses abstraksi.

1. **Parameterized Abstract Data Type**
* **Abstract Data Tipe**

Sebuah tipe data abstrak adalah tipe data yang ditentukan oleh pengguna yang memenuhi 2 kondisi dibawah :

* Representasi dari, dan operasi pada, objek dari tipe yang di definisi pada unit sintaktik tunggal
* Representasi dari objek sebuah tipe disembunyikan dari unit program yang menggunakanobjek tersebut, jadi operasi yang mungkin hanya yang di sediakan dalam definisi tipe tersebut.
* **Parameterized Abstract Data Types**

Tipe Data Abstrak Yang Di Parameterisasi memungkinkan pembuatan sebuah Tipe Data Abstrak yang bisa menyimpan segala elements(diantara benda lain)

* Hanya sebuah isu untuk bahasa statik
* C++, Ada, Java 5.0, dan C# 2005 menyediakan dukungan untuk tipe data abstrak yang di parameterisasi

Ini adalah implementasi Tipe Data Abstrak menggunakan Python list:

class Stack :

 def \_\_init\_\_(self):

 self.items = []

 def push(self, item):

 self.items.append(item)

 def pop(self):

 return self.items.pop()

 def is\_empty(self):

 return (self.items == [])

Sebuah objek Stack memuat atribut bernama items yang merupakan list dari items dalam stack. Metode inisialisasi tersebut mencantumkan items kedalam list yang kosong.Untuk memasukkan item baru kedalam list, push menambahkannya kepada items. Untuk memasukkan dan mengeluarkan sebuah item, pop menggunakan sebuah metode list yang homonim untuk menghilangkan dan mengembalikan item terakhir kedalam list.Akhirnya untuk mengecek apakah list kosong, is\_empty membandingkan items ke list yang kosong.

Para ilmuwan komputer menggunakan metafor ini untuk mendeskripsikan sebuah potongan kecil kode yang menyembunyikan detil dari implementasi dan menyediakan sebuah pelayanan yang lebih simpel dan standar.

1. **Naming Encapsulation**

Digunakan untuk merujuk kepada salah satu dari dua konsep terkait tetapi berbeda dan digunakan juga untuk mengkombinasi Mekanisme bahasa untuk membatasi akses langsung ke beberapa komponen objek dan sebuah konstruksi bahasa yang memfasilitasi bundling data dengan metode (atau fungsi lainnya) yang beroperasi pada data tersebut.

class C(object):

def \_\_init\_\_(self):

self.a = 123 # OK to access directly

self.\_a = 123 # should be considered private

self.\_\_a = 123 # considered private, name mangled

>>> c = C()

>>> c.a

123

>>> c.\_a

123

>>> c.\_\_a

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AttributeError: 'C' object has no attribute '\_\_a'

>>> c.\_C\_\_a

123

Sebuah underscore menunjukkan kepada pengguna dari kelas yang atribut harus dipertimbangkan pribadi ke kelas, dan tidak boleh diakses secara langsung.Sebuah underscore ganda menunjukkan yang sama, bagaimanapun, Python akan mangle nama atribut untuk mencoba untuk menyembunyikannya.

1. **Encapsulation Construct**

Encapsulation dalam python adalah pembatasan akses terhadap method dan variabel untuk mencegah data termodifikasi secara tidak sengaja.

**4.1 Private method**

Sebuah method yang hanya dapat diakses pada kelasnya sendiri dengan cara dimulai dengan dua underscore.

Contoh:

*#!/usr/bin/env python*

class Car:

def \_\_init\_\_(self):

 self.\_\_updateSoftware()

def drive(self):

print 'driving'

def \_\_updateSoftware(self):

print 'updating software'

redcar = Car()

redcar.drive()

*#redcar.\_\_updateSoftware() not accesible from object.*

Output :

updating software

driving

* 1. **Private variabel**

Sebuah variabel private hanya dapat diubah didalam kelas method dan tidak dapat diubah luar kelas dimana ia didefinisikan.

Contoh :

*#!/usr/bin/env python*

**class** Car:

 \_\_maxspeed = 0

 \_\_name = ""

**def** \_\_init\_\_(self):

 self.\_\_maxspeed = 200

 self.\_\_name = "Supercar"

**def** drive(self):

**print** 'driving. maxspeed ' + str(self.\_\_maxspeed)

redcar = Car()

redcar.drive()

redcar.\_\_maxspeed = 10 *# will not change variable because its private*

redcar.drive()

1. **Example of Abstraction**

# Import library os untuk membersihkan layar console

import os

# Class stack

class Stack:

    def \_\_init\_\_(self):

        self.items = []

    # Memeriksa apakah stack kosong

    def isEmpty(self):

        return self.items == []

    # Menambah objek/data ke dalam stack

    def push(self, item):

        self.items.append(item)

    # Mengeluarkan data dari stack

    def pop(self):

        return self.items.pop()

    # Menampilkan objek terakhir dari stack

    def peek(self):

        return self.items[len(self.items)-1]

    # Mehitung panjang stack

    def size(self):

        return len(self.items)

    # Menu dari aplikasi

    def mainmenu(self):

        pilih = "y"

        while (pilih == "y"):

            os.system("clear")

print("=========================")

            print("|  Menu aplikasi stack  |")

print("===================== ====")

            print("1. Push objek")

            print("2. Pop objek")

            print("3. Cek Empty")

            print("4. Tampil objek terakhir")

            print("5. Panjang dari stack")

print("=========================")

            pilihan=str(input(("Silakan masukan pilihan anda: ")))

            if(pilihan=="1"):

                os.system("clear")

                obj = str(input("Masukan objek yang ingin anda tambahkan: "))

                self.push(obj)

                print("Object "+obj+" telah ditambahkan")

                x = raw\_input("")

            elif(pilihan=="2"):

                os.system("clear")

                print("Objek "+self.pop()+" dihapus")

                x = raw\_input("")

            elif(pilihan=="3"):

                os.system("clear")

                print(self.isEmpty())

                x = raw\_input("")

            elif(pilihan=="4"):

                os.system("clear")

                print("Objek terakhir: "+self.peek())

                x = raw\_input("")

            elif(pilihan=="5"):

                os.system("clear")

                print("Panjang dari stack adalah: "+str(self.size()))

                x = raw\_input("")

            else:

                pilih="n"

if \_\_name\_\_ == "\_\_main\_\_":

    s=Stack()

    s.mainmenu()

**BAB 8**

**Object-Oriented Programming**

1. **Pengertian OOP (Object Oriented Programming)**

 OOP (Object Oriented Programming) adalah suatu metode pemrograman yang berorientasi kepada objek. Tujuan dari OOP diciptakan adalah untuk mempermudah pengembangan program dengan cara mengikuti model yang telah ada di kehidupan sehari-hari. Jadi setiap bagian dari suatu permasalahan adalah objek, nah objek itu sendiri merupakan gabungan dari beberapa objek yang lebih kecil lagi. Saya ambil contoh Pesawat, Pesawat adalah sebuah objek. Pesawat itu sendiri terbentuk dari beberapa objek yang lebih kecil lagi seperti mesin, roda, baling-baling, kursi, dll. Pesawat sebagai objek yang terbentuk dari objek-objek yang lebih kecil saling berhubungan, berinteraksi, berkomunikasi dan saling mengirim pesan kepada objek-objek yang lainnya. Begitu juga dengan program, sebuah objek yang besar dibentuk dari beberapa objek yang lebih kecil, objek-objek itu saling berkomunikasi, dan saling berkirim pesan kepada objek yang lain.

1. **Konsep OOP (Object Oriented Programming)**
	1. **Kelas Abstrak (Class Abstraksi)**
* Kelas merupakan deskripsi abstrak informasi dan tingkah laku dari sekumpulan data.
* Kelas dapat diilustrasikan sebagai suatu cetak biru(blueprint) atau prototipe yang digunakan untuk menciptakan objek.
* Kelas merupakan tipe data bagi objek yang mengenkapsulasi data dan operasi pada data dalam suatu unit tunggal.
* Kelas mendefinisikan suatu struktur yang terdiri atas data kelas (data field), prosedur atau fungsi (method), dan sifat kelas (property).
	1. **Enkapsulasi (encapsulation)**
* Istilah enkapsulasi sebenarnya adalah kombinasi data dan fungsionalitas dalam sebuah unit tunggal sebagai bentuk untuk menyembunyikan detail informasi.
* Proses enkapsulasi memudahkan kita untuk menggunakan sebuah objek dari suatu kelas karena kita tidak perlu mengetahui segala hal secara rinci.
* Enkapsulasi menekankan pada antarmuka suatu kelas, atau dengan kata lain bagaimana menggunakan objek kelas tertentu.
* Contoh: kelas mobil menyediakan antarmuka fungsi untuk menjalankan mobil tersebut, tanpa kita perlu tahu komposisi bahan bakar, udara dan kalor yang diperlukan untuk proses tersebut.
	1. **Pewarisan (Inheritance)**
* Kita dapat mendefinisikan suatu kelas baru dengan mewarisi sifat dari kelas lain yang sudah ada.
* Penurunan sifat ini bisa dilakukan secara bertingkattingkat, sehingga semakin ke bawah kelas tersebut menjadi semakin spesifik.
* Sub kelas memungkinkan kita untuk melakukan spesifikasi detail dan perilaku khusus dari kelas supernya.
* Dengan konsep pewarisan, seorang programmer dapat menggunakan kode yang telah ditulisnya pada kelas super berulang kali pada kelas-kelas turunannya tanpa harus menulis ulang semua kodekode itu.
	1. **Polimorfisme (polymorphism)**
* Polimorfisme merupakan kemampuan objekobjek yang berbeda kelas namun terkait dalam pewarisan untuk merespon secara berbeda terhadap suatu pesan yang sama.
* Polimorfisme juga dapat dikatakan kemampuan sebuah objek untuk memutuskan method mana yang akan diterapkan padanya, tergantung letak objek tersebut pada jenjang pewarisan.
* Method overriding.
* Method name overloading.
1. **Karakteristik OOP (Object Oriented Programming)**

Semua adalah objek.

* Komputasi dilakukan dengan komunikasi antar objek. Setiap objek berkomunikasi dengan objek yang lain melalui pengiriman dan penerimaan pesan.
* Sebuah pesan merupakan permintaan atas sekumpulan aksi dengan semua argumen yang diperlukan untuk menyelesaikan suatu tugas tertentu.
* Setiap objek memiliki memori sendiri, yang dapat terdiri dari objek-objek lainnya.
* Setiap objek adalah wakil atau representasi dari suatu kelas. Sebuah kelas dapat mewakili sekelompok objek yang sama.
* Kelas merupakan kumpulan tingkah laku yang berkaitan dengan suatu objek. Jadi, semua objek yang merupakan wakil dari kelas yang sama dapat melakukan aksi yang sama pula.
* Kelas-kelas diorganisasikan ke dalam struktur pohon yang berakar tunggal, yang dinamakan dengan jenjang pewarisan (inheritance hierarchy).
* Setiap objek pada umumnya memiliki tiga sifat, yaitu keadaan, operasi dan identitas objek.
* Operasi merupakan tindakan yang dapat dilakukan oleh sebuah objek.
* Keadaan objek merupakan koleksi dari seluruh informasi yang dimiliki oleh objek pada suatu saat.
1. **Semuanya Merupakan Objek**

Untuk apakah kata kunci class digunakan, tepatnya? Seperti sepupunya def berdasarkan fungsinya, berhubungan dengan definisi dari benda.Sembari def digunakan untuk mendefinisikan fungsi, class digunakan untuk mendefinisikan kelas. Dan apa itu class ? Simpelnya pengelompokan data dan fungsi secara logis (yang sering disebut “metode” dalam kelas).

Apa yang kita maksud dengan “pengelompokan logika” ? Sebuah kelas dapat memuat data apapun yang kita mau, dan bisa memiliki fungsi(metode) apapun sesuka kita. Daripada menaruh segala sesuatu yang berbeda dalam "kelas" secara asal, kita mencoba untuk membuat kelas dimana ada hubungan yang logis antar benda. Sering kali kelas berdasarkan pada objek (seperti Customer atau Product). Pada lain waktu, kelas berdasarkan pada konsep, seperti HTTPRequest atau Owner.

Walaupun, kelas-kelas adalah teknik modeling; sebuah cara berpikir tentang program. Ketika anda berpikir dan mengimplementasikan sistem anda dalam cara ini, anda dikatakan sedang melakukan Object-Oriented Programming. “Kelas-kelas” dan “Objek-objek” adalah kata yang sering digunakan bergantian, tetapi keduanya bukan hal yang sama. Memahami apa yang membuat mereka berbeda adalah kunci untuk memahami apakah mereka sebenarnya dan cara kerja mereka.

1. **Semuanya Memiliki Kelas?**

Kelas-kelas bisa dianggap sebagai *cetak biru untuk menciptakan objek*. Ketika mendefinisikan sebuah kelas customer menggunakan kata kunci class. Sebenarnya kita belum benar benar menciptakan customer. Tetapi, apa yang telah kita ciptakan adalah sebuah instruksi manual untuk mengkonstruksi objek “customer”. Mari lihat contoh berikut:

**classCustomer**(object):

*"""A customer of ABC Bank with a checking account. Customers have the*

 *following properties:*

 *Attributes:*

 *name: A string representing the customer's name.*

 *balance: A float tracking the current balance of the customer's account.*

 *"""*

**def** \_\_init\_\_(self, name, balance=0.0):

*"""Return a Customer object whose name is \*name\* and starting*

 *balance is \*balance\*."""*

 self.name = name

 self.balance = balance

**def** withdraw(self, amount):

*"""Return the balance remaining after withdrawing \*amount\**

 *dollars."""*

**if** amount > self.balance:

**raiseRuntimeError**('Amount greater than available balance.')

 self.balance -= amount

**return** self.balance

**def** deposit(self, amount):

*"""Return the balance remaining after depositing \*amount\**

 *dollars."""*

 self.balance += amount

**return** self.balance

Baris class Customer(object) tidak menciptakan sebuah customer baru. Hanya karena kita telah mendefinisikan sebuah Customer tidak berarti kita telah menciptakannya; kita hanya mengarisbesarkan cetak biru untuk menciptakan objek Customer. Untuk melakukannya kita memanggil metode \_\_init\_\_ dengan beberapa argument (minus self, yang mana akan kita pelajari sebentar lagi).

Jadi untuk menggunakan "cetak biru" yang kita ciptakan dengan mendefinisikan class Customer(yang mana digunakan untuk menciptakan objek Customer), kita memanggil nama class tersebut sama seperti fungsi: jeff = Customer('Jeff Knupp', 1000.0). Baris ini hanya berbunyi "gunakan cetak biru Customeruntuk menciptakan sebuah objek baru untuk saya, yang mana saya sebut sebagaijeff."

Objek jeff ini, dikenal sebagai instance, adalah bentuk realisasi dari class Customer . Sebelum kita memanggil Customer(), tidak ada objek Customer . Tentu saja kita bisa membuat objek Customer sebanyak mungkin yang kita mau. Tapi hanya ada 1 kelas Customer , tidak peduli berapa banyak instansi dari kelas yang kita buat.

1. **Penggunaan Self**

Jadi ada apa dengan parameter self terhadap semua metode Customer ? Apakah itu? Tentu saja instansinya ! Dengan cara lain, sebuah metode seperti withdraw mendefinisikan sebuah instruksi untuk mengambil uang dari akun customer abstrak. Memanggil jeff.withdraw(100.0) menaruh semua instruksi tersebut untuk digunakan di instansi jeff .

Jadi apa yang kita sebut def withdraw(self, amount):, seperti, "inilah cara anda menarik uang dari class customer (yang mana kita sebut self) dan sebuah figur dolar (yang mana kita sebut amount). Self adalah instasi dari Customer yang withdraw sedang panggil. Tentu saja bukan saya yang membuat analogi tersebut.jeff.withdraw(100.0) hanyalah singkatan Customer.withdraw(jeff, 100.0), kode yang sangat valid (jika tidak sering dilihat).

1. **Penggunaan \_\_init\_\_**

Self mungkin masuk akal untuk metode lain, tapi bagaimana dengan \_\_init\_\_? Ketika kita memanggil \_\_init\_\_, kita sedang dalam proses menciptakan objek, jadi bagaimana bisa sudah tersedia self? Python memperbolehkan kita untuk memperluas corak self sampai objek dikonstruksi, walaupun tidak sepenuhnya cocok. Imaginasikan jeff = Customer('Jeff Knupp', 1000.0) sama dengan memanggil jeff = Customer(jeff, 'Jeff Knupp', 1000.0); jeff yang dilewatkan juga menghasilkan nilai.

Oleh karena itu mengapa ketika kita memanggil \_\_init\_\_, kita menginisialisasikan objek dengan cara seperti self.name = name. Ingat, karena self adalah instansi, ini sama dengan jeff.name = name, yang juga sama dengan jeff.name = 'Jeff Knupp. Begitupula, self.balance = balance sama dengan jeff.balance = 1000.0. Setelah 2 baris ini, kita menganggap objek Customer terinisialisasi dan siap dipakai. Setelah \_\_init\_\_ selesai, pemanggil dapat mengasumsikan bahwa objek siap dipakai. Setelah jeff = Customer('Jeff Knupp', 1000.0), kita mulai membuat deposit dan panggilan withdraw terhadap jeff; jeff yang merupakan objek yang terinisialisasi sepenuhnya. Imaginasikan untuk sementara kita telah mendefinisikankelas Customer dengan sedikit berbeda:

**classCustomer**(object):

*"""A customer of ABC Bank with a checking account. Customers have the*

 *following properties:*

 *Attributes:*

 *name: A string representing the customer's name.*

 *balance: A float tracking the current balance of the customer's account.*

 *"""*

**def** \_\_init\_\_(self, name):

*"""Return a Customer object whose name is \*name\*."""*

 self.name = name

**def** set\_balance(self, balance=0.0):

*"""Set the customer's starting balance."""*

 self.balance = balance

**def** withdraw(self, amount):

*"""Return the balance remaining after withdrawing \*amount\**

 *dollars."""*

**if** amount > self.balance:

**raiseRuntimeError**('Amount greater than available balance.')

 self.balance -= amount

**return** self.balance

**def** deposit(self, amount):

*"""Return the balance remaining after depositing \*amount\**

 *dollars."""*

 self.balance += amount

**return** self.balance

Ini mungkin terlihat seperti alternatif yang masuk akal; kita hanya perlu memanggil set\_balance sebelum kita memulai menggunakan instansi. Walaupun tidak ada cara untuk berkomunikasi kepada pemanggilnya. Walaupun kita mendokumennya secara ekstensif kita tidak bisa memaksa pemanggilnya memanggil jeff.set\_balance(1000.0) sebelum memanggil jeff.withdraw(100.0). Karena instansi jeff tidak memiliki atribut yang seimbang sampai jeff.set\_balance dipanggil, ini berarti bahwa objek belum sepenuhnya terinisialisasi.

Peraturannya adalah jangan perkenalkan atribut baru diluar metode \_\_init\_\_, kecuali anda telah memberikan pemanggilnya sebuah objek yang belum terinisialisasi sepenuhnya. Ada beberapa pengecualin tentunya, tapi itu adalah sebuah prinsip yang baik untuk diingat. Ini adalah bagian dari konsep yang lebih besar dari konsistensi objek: tidak boleh ada seri pemanggilan metode yang bisa mengakibatkan objek memasuki sebuah situasi yang tidak masuk akal.

Invariants (seperti, “seimbang harus selalu angka yang tidak negatif”) harus menampung keduanya ketika sebuah metode dimasuki dan keluar. Seharusnya tidak mungkin untuk sebuah objek masuk ke sebuah situasi yang invalid hanya dengan memanggil metodenya. Sebuah objek harus bermulai dalam kondisi yang valid, dimana oleh karena itu penting untuk menginisialisasikan segalanya dalam metode \_\_init\_\_.

1. **Instance Attributes and Methods**

Fungsi didefinisikan dalam kelas disebut "metod". Metod memiliki akses ke semua data yang terdapat pada instance dari objek; mereka dapat mengakses dan mengubah apa yang sebelumnya ditetapkan pada self (diri dari objek itu sendiri). Karena mereka menggunakan self, mereka membutuhkan sebuah instance dari kelas agar dapat digunakan. Untuk alasan ini, mereka sering disebut sebagai "instance method". Jika ada "instance metode", maka pasti ada jenis lain dari metode juga, akan tapi metod-method tersebut sedikit lebih bersifat esoteris.

1. **Static Methods**

*Class attributes* merupakan atribut yang ditetapkan pada class-level, yang bertentangan dengan instance-level. atribut normal diperkenalkan dalam metod \_\_init\_\_, tetapi beberapa atribut dari suatu kelas menyimpan semua instances dalam semua kasus. Sebagai contoh, perhatikan definisi dari objek Car berikut:

**classCar**(object):

 wheels = 4

**def** \_\_init\_\_(self, make, model):

 self.make = make

 self.model = model

mustang = Car('Ford', 'Mustang')

**print** mustang.wheels

*# 4*

**print** Car.wheels

*# 4*

Objek car selalu memiliki empat roda, terlepas dari model objek car-nya. Instance method dapat mengakses atribut ini dengan cara yang sama ketika mereka mengakses atribut biasa: melalui self (yaitu self.wheels).

Ada kelas metode yang disebut metode statis, yang tidak memiliki akses ke self. Sama seperti atribut kelas, mereka adalah metode yang bekerja tanpa memerlukan sebuah instance. Karena instance selalu direferensikan melalui self, metode statis tidak memiliki parameter sendiri.

Berikut ini merupakan static methode yang valid dari class car:

**classCar**(object):

 ...

**def** make\_car\_sound():

**print** 'VRooooommmm!'

Tidak peduli apa jenis car yang dimiliki, sebuah car selalu mengelurakan suara yang sama. Untuk membuat lebih jelas bahwa metode ini tidak harus menerima instance sebagai parameter pertama (self yaitu pada metode "normal"), dekorator @staticmethod digunakan, mengubah definisi method kita menjadi:

**classCar**(object):

 ...

 @staticmethod

**def** make\_car\_sound():

**print** 'VRooooommmm!'

1. **Class Methods**

Sebuah variasi dari static method adalah *class method*. Dibandingkan menerima sebuah instance menjadi sebuah parameter pertamanya, class method melewatkan instance pada class. Class method juga didefinisikan menggunakan dekorator:

**classVehicle**(object):

 ...

 @classmethod

**def** is\_motorcycle(cls):

**return** cls.wheels == 2

1. **Inheritance**

Pemrograman berorientasi objek berguna sebagai alat pemodelan, dan sangat luas implementasinya ketika konsep warisan (inheritance) diperkenalkan. Inherticance adalah proses dimana kelas "child" berasal data dan perilaku dari kelas "parent". Contoh pasti akan membantu kami di sini.

Bayangkan kita menjalankan sebuah dealer mobil yang menjual semua jenis kendaraan, dari sepeda motor sampai truk. Kami menetapkan untuk tidak bersaing harga. Secara khusus, bagaimana kita menentukan harga kendaraan yang terdapat pada lapak kami: $ 5,000 x jumlah roda yang dimemiliki kendaraan. Kami juga menerima untuk membeli kembali kendaraan yang sebelumnya telah kami jual. Kami menawarkan flat rate - 10% dari jarak yang telah ditempuh kendaraan dalam mil. Untuk truk, $ 10.000. Untuk mobil, $ 8.000. Untuk sepeda motor, $ 4.000.

Jika kita ingin membuat sistem penjualan untuk dealer kami menggunakan teknik berorientasi objek, bagaimana kita akan melakukannya? Seperti apa perwujudan objeknya? Kita mungkin memiliki kelas Sale, kelas Pelanggan, kelas Inventarisasi, dan sebagainya, tapi kita akan pasti memiliki kelas mobil, kelas truk, dan kelas Motor. Seperti apa perwujudan dari kelas tersebut? Menggunakan apa yang telah kita pelajari, berikut adalah kemungkinan implementasi dari kelas Mobil:

**classCar**(object):

*"""A car for sale by Jeffco Car Dealership.*

 *Attributes:*

 *wheels: An integer representing the number of wheels the car has.*

 *miles: The integral number of miles driven on the car.*

 *make: The make of the car as a string.*

 *model: The model of the car as a string.*

 *year: The integral year the car was built.*

 *sold\_on: The date the vehicle was sold.*

 *"""*

**def** \_\_init\_\_(self, wheels, miles, make, model, year, sold\_on):

*"""Return a new Car object."""*

 self.wheels = wheels

 self.miles = miles

 self.make = make

 self.model = model

 self.year = year

 self.sold\_on = sold\_on

**def** sale\_price(self):

*"""Return the sale price for this car as a float amount."""*

**if** self.sold\_on **isnot** None:

**return** 0.0 *# Already sold*

**return** 5000.0 \* self.wheels

**def** purchase\_price(self):

*"""Return the price for which we would pay to purchase thecar."""*

**if** self.sold\_on **is** None:

**return** 0.0 *# Not yet sold*

**return** 8000 - (.10 \* self.miles)

 ...

Sekarang kita punya kelas mobil, mungkin kita juga harus membuat kelas Truck. Pemuatan kelas truk akan mengikuti pola yang sama dengan pembuatan kelas untukmobil:

**classTruck**(object):

*"""A truck for sale by Jeffco Car Dealership.*

 *Attributes:*

 *wheels: An integer representing the number of wheels the truck has.*

 *miles: The integral number of miles driven on the truck.*

 *make: The make of the truck as a string.*

 *model: The model of the truck as a string.*

 *year: The integral year the truck was built.*

 *sold\_on: The date the vehicle was sold.*

 *"""*

**def** \_\_init\_\_(self, wheels, miles, make, model, year, sold\_on):

*"""Return a new Truck object."""*

 self.wheels = wheels

 self.miles = miles

 self.make = make

 self.model = model

 self.year = year

 self.sold\_on = sold\_on

**def** sale\_price(self):

*"""Return the sale price for this truck as a float amount."""*

**if** self.sold\_on **isnot** None:

**return** 0.0 *# Already sold*

**return** 5000.0 \* self.wheels

**def** purchase\_price(self):

*"""Return the price for which we would pay to purchase the truck."""*

**if** self.sold\_on **is** None:

**return** 0.0 *# Not yet sold*

**return** 10000 - (.10 \* self.miles)

 ...

**BAB 9**

**CONCURRENCY**

**FUNTIONAL PROGRAMMING LANGUAGES**

Pada Python tidak ada functional programming. Functional programming adalah model pemograman yang model komputasinya merupakan evaluasi dari ekspresi. Desain bahasa imperatif berdasarkan arsitektur von Neumann. Desain dari bahasa fungsional berasal dari fungsi matematika.

1. **Fungsi Matematika**

Pemetaan anggota dari sebuah kelompok yang disebut domain ke kelompok yang lain yang disebut range (daerah hasil). Ekspresi lambda (λ) yang mengspesifikasi parameternya dan pemetaan sebuah fungsi. Ekspresi lambda mendeskripsikan fungsi yang tidak memiliki nama.

1. **Higher-order Function**

Disebut juga functional form. Merupakan salah satu yang mengambil fungsi sebagai parameter atau hasil sebuah fungsi sebagai hasil atau keduanya.

1. **Purity**

Bahasa yang melarang efek samping. Beberapa bahasa fungsional mempernolehkan ekspresi untuk yield action untuk menghasilkan return value.

* 1. **Immutable Data**

Program fungsional purely biasanya beroperasi pada immutable data. Selain mengubah hasil yang sudah ada, hasil copy sudah terbuat dan yang aslinya tetap masih ada.

* 1. **Referential Transparency**

Hasil komputasi pure hasilnya sama setiap mereka dipanggil.

**3.3 Lazy Evaluation**

Menghindari komputasi yang tidak sesuai dan memperbolehkan struktur data yang tak terhingga supaya dikenal.

1. **Perbedaan Functional dan Imperative Language**

Imperative Language:

* Eksekusinya efisien
* Sintaks dan semantiks kompleks
* Concurrency didesain programmer

Functional Language:

* Sintaks dan semantiks simpel
* Eksekusi kurang efisien
* Program bisa jadi concurrent secara otomatis

**BAB 10**

**EXCEPTION HANDLING AND EVENT HANDLING**

1. **Exception Handling**

**1.1 Introducing Exception Handling**

Exception Handling adalah sebuah event yang terjadi ketika program menemui kesalahan pada saat instruksi program dijalankan.

Syntax untuk Exception Handling

try:

 You do your operations here;

 ......................

except *ExceptionI*:

 If there is ExceptionI, then execute this block.

except *ExceptionII*:

 If there is ExceptionII, then execute this block.

 ......................

else:

 If there is no exception then execute this block.

Contoh :

#!/usr/bin/python

try:

 fh = open("testfile","w")

 fh.write("This is my test file for exception handling!!")

exceptIOError:

print"Error: can\'t find file or read data"

else:

print"Written content in the file successfully"

 fh.close()

Output :

Written content in the file successfully

**1.2 Event Handling**

Event Handling adalah suatu metode untuk menangani sebuah event/aksi yang diberikan pengguna kepada suatu komponen.

Contoh : from axel import Event

event = Event()

def on\_event(\*args, \*\*kwargs):

 return (args, kwargs)

event += on\_event # handler registration

print(event(10, 20, y=30))

>> ((True, ((10, 20), {'y': 30}), <function on\_event at 0x00BAA270>),)

event -= on\_event # handler is unregistered

print(event(10, 20, y=30))

>> None

class Mouse(object):

 def \_\_init\_\_(self):

 self.click = Event(self)

 self.click += self.on\_click

 def on\_click(self, sender, \*args, \*\*kwargs):

 assert isinstance(sender, Mouse), 'Wrong sender'

 return (args, kwargs)

mouse = Mouse()

print(mouse.click(10, 20))

>> ((True, ((10, 20), {})

>><bound method Mouse.on\_click of <\_\_main\_\_.Mouse object at 0x00B6F470>>),)

mouse.click -= mouse.on\_click

print(mouse.click(10, 20))

>> None

Fitur-fitur axel sebagai library python dalam event handling :

1. Handler dapat menerima sender sebagai argument pertama
2. Handler dapat dijalankan pada thread yang sama maupun berbeda
3. Alokasi waktu untuk menjalankan handler dapat dikendalikan
4. Handler yang dijalankan dalam satu waktu dapat ditentukan jumlahnya
5. Kemudahan dalam melakukan traceback jika terjadi error pada handler
6. **Syntax Errors**

Syntax error atau yang dikenal juga dengan parsing error adalah mungkin complain yang paling umum ketika kamu belajar python

**>>>whileTrue**print('Hello world')

 File "<stdin>", line 1

**whileTrue**print('Hello world')

^

SyntaxError: invalid syntax

Pengulangan parser yang menyinggung baris dan menampilkan arrow kecil yang mengarah kepada poin paling awal dari baris dimana terdapat error. Error disebabkan oleh token preceding. Seperti contoh , error yang terdeteksi pada fungsi print () karena menghilangnya ; . nama file dan baris juga di print sehingga dapat diketahui dimana harus mencarinya pada script.

1. **Exceptions**

Bahkan ketika sebuah statement atau ekspresi memiliki syntax yang benar itu juga dapat menyebabkan error ketika sebuah usaha untuk menjalankannya. Error dideteksi ketika penjalanan program yang disebut dengan exception dan sifatnya tidak fatal. Kebanyakn exception tidak ditangani oleh program namun hasil dari pesan error ditampilkan disini.

**>>>**10\* (1/0)

Traceback (most recent call last):

 File "<stdin>", line 1, in <module>

ZeroDivisionError: division by zero

**>>>**4+ spam\*3

Traceback (most recent call last):

 File "<stdin>", line 1, in <module>

NameError: name 'spam' is not defined

**>>>**'2'+2

Traceback (most recent call last):

 File "<stdin>", line 1, in <module>

TypeError: Can't convert 'int' object to str implicitly

Baris terakhir pada pesan error mengindikasikan apa yang terjadi. Exception dapat masuk melalui tipe yang berbeda dan tipe berbeda tersebut adalah bagian dari pesan. Contoh diatas adalah zerodivisionerror, namerror,typeerror. Sebuah string yang tertulis dalam exception yang terjadi. Built exception seperti ini adalah benar. Tapi membutuhkan exception yang didefinisikan oleh user. Exception standar dibuat dari identifier. Baris terakhir menyediakan detil mengenai tipe exception dan apa yang menyebabkannya.

Bagian preceding dari pesan error memperlihatkan konteks dimana exception terjadi. Dalam bentuk sebuah traceback stak. Pada umumnya ini mengandung sebuah traceback stack list baris. Namun tidak menunjukkan baris baca dari input standar

Try statement bekerja sebagai berikut.

* Pertama try clause diantara try dan except di jalankan
* Jika tidak terjadi exception. Maka exception clause akan di skup dan eksekusi dari try statement selesai.
* Jika exception terjadi ketika eksekusi dari try clause. Sisa dari try clause akan di skip. Dan tipe dari exception setelah kata except akan dijalankan yang akan terus berlanjut sampai try statement.
* Jika sebuah exception terjadi dimana tidak cocok dengan nama exception pada except clause maka itu akan di oper keluar try statement. Jika tidak ada penanganan yang ditemukan maka akan menjadi unhandled exception dan berhenti dengan menampilkan pesan seperti diatas.

Sebuah statement mungkin memiliki lebih dari 1 except clause untuk secara khusus mengurusi exception yang berbeda. Satu handler akan di eksekusi. Dan handler hanya akan mengurusi exception yang terjadi pada try clause bukan dari handler lain dari try statement. Sebuah except clause mungkin menamai beberapa exception sebagai parenthesized tuple.

...**except** (RuntimeError, TypeError, NameError):

...**pass**

Sebuah kelas dalam except clause cocok dengan exception jika memiliki kelas yang sama atau dasar kelas. Daripadanya sebuah except clause list di turunkan dari kelas yang tidak cocok dengan base class . contoh :

**classB**(Exception):

**pass**

**classC**(B):

**pass**

**classD**(C):

**pass**

**for** cls **in** [B, C, D]:

**try**:

**raise** cls()

**except** D:

print("D")

**except** C:

print("C")

**except** B:

print("B")

Akhir dari except clause mungkin menghilankan nama exception untuk berperan sebagai wildcard. Gunakan ini dengan hati-hati karena sangat mudah untuk terjadi programming error pada cara ini. Ini juga dapat digunakan untuk mengeprint pesan error dan menaikkan kembali exception

**importsys**

**try**:

f=open('myfile.txt')

s=f.readline()

i=int(s.strip())

**except**OSError**as**err:

print("OS error: *{0}*".format(err))

**except**ValueError:

print("Could not convert data to an integer.")

**except**:

print("Unexpected error:",sys.exc\_info()[0])

**raise**

Statement *try…except* memiliki sebuah klausa opsional *else*, yang dimana ketika penggunaannya, harus mengikuti semua klausa *except*. Hal ini berguna untuk kode yang harus dijalankan jika klausa *try* tidak meningkatkan exception. Sebagai contoh:

**for**arg**in**sys.argv[1:]:

**try**:

f=open(arg,'r')

**except**IOError:

print('cannot open',arg)

**else**:

print(arg,'has',len(f.readlines()),'lines')

f.close()

Penggunaan klausa *else* lebih baik daripada menambahkan kode tambahan untuk klausa *try,* Karena ini menghindari kejadian yang tanpa sengaja menangkap *exception* yang tidak support oleh kode dilindungi oleh statement *try ... except*.

Ketika exception terjadi, itu mungkin memiliki nilai yang terkait, yang juga dikenal sebagai exception argument. Kehadiran dan jenis argumen tergantung pada jenis exception.

Sebuah klausa *except* dapat menentukan variabel setelah nama exception. Variabel terikat ke sebuah *exception instance* dengan argumen yang tersimpan di ininstance.args. Untuk kenyamanan, exception instance mendefinisikan \_\_str \_\_ () sehingga argumen dapat dicetak langsung tanpa harus mereferensi .args. Selain itu, juga dapat instantiate sebuah exception terlebih dahulu sebelum raising dan menambahkan atribut apapun untuk itu seperti yang diinginkan.

**>>>try**:

**... raise**Exception('spam','eggs')

**... except**Exception**as**inst:

**...** print(type(inst))*# the exception instance*

**...** print(inst.args)*# arguments stored in .args*

**...** print(inst)*# \_\_str\_\_ allows args to be printed directly,*

**...** *# but may be overridden in exception subclasses*

**...** x,y=inst.args*# unpack args*

**...** print('x =',x)

**...** print('y =',y)

**...**

<class 'Exception'>

('spam', 'eggs')

('spam', 'eggs')

x = spam

y = eggs

Jika sebuah exception memiliki argumen, mereka akan dicetak sebagai bagian terakhir ('detail') dari pesan untuk exception yang tidak ter-handle.

Penangan exception tidak hanya menangani exception jika exception terjadi dengan segera dalam klausa *try*, tetapi juga saat exception terjadi di dalam fungsi yang dipanggil (bahkan secara tidak langsung) dalam klausa *try*. Sebagai contoh:

**>>>def**this\_fails():

**...** x=1/0

**...**

**>>>try**:

**...** this\_fails()

**... except**ZeroDivisionError**as**err:

**...** print('Handling run-time error:',err)

**...**

Handling run-time error: division by zero

1. **Raising Exceptions**

Sebuah statement raise memungkinkan seorang programmer untuk memaksakan sebuah exception tertentu untuk terjadi. Sebagai contoh:

**>>>raise**NameError('HiThere')

Traceback (most recent call last):

 File "<stdin>", line 1, in <module>

NameError: HiThere

Satu-satunya argumen untuk raise menunjukkan exception untuk di-raise. Ini menjadi salah contoh exception atau exception class (kelas yang berasal dari Exception). Jika exception class dilewatkan, maka akan secara implisit dipakai dengan memanggil konstruktor tanpa argumen:

**raise**ValueError*# shorthand for 'raise ValueError()'*

Jika kita perlu untuk menentukan apakah sebuah exception di-raise tetapi tidak berniat untuk melakukan handle terhadapnya, sebuah bentuk sederhana dari statement raise memungkinkan kita untuk melakukan re-raise terhadap exception.

**>>>try**:

**... raise**NameError('HiThere')

**... except**NameError:

**...** print('An exception flew by!')

**... raise**

**...**

An exception flew by!

Traceback (most recent call last):

 File "<stdin>", line 2, in <module>

NameError: HiThere

1. **User-defined Exceptions**

Program dapat menamai exception mereka dengan membuat sebuah kelas exception baru. Exception biasanya diturunkan dari kelas exception. Yaitu secara langsung maupun tak langsung kelas exception dapat didefinisikan untuk melakukan apa yang dapat dilakukan kelas lain, tetapi biasanya dibuat lebih simple, atau sering kali menawarkan sebuah angka dari atribut yang mengizinkan informasi mengenai error, di ekstrak oleh handler untuk exception tersebut. Ketika membuat sebuah modul yang dapat menambah beberapa error istimewa. Sebuah latihan biasa untuk membuat kelas dasar dari exception diartikan dari modul itu, dan subclass yang membuat kelas exception spesifik untuk kondisi error yang berbeda.

**classError**(Exception):

*"""Base class for exceptions in this module."""*

**pass**

**classInputError**(Error):

*"""Exception raised for errors in the input.*

 *Attributes:*

 *expression -- input expression in which the error occurred*

 *message -- explanation of the error*

 *"""*

**def**\_\_init\_\_(self, expression, message):

self.expression = expression

self.message = message

**classTransitionError**(Error):

*"""Raised when an operation attempts a state transition that's not*

 *allowed.*

 *Attributes:*

 *previous -- state at beginning of transition*

 *next -- attempted new state*

 *message -- explanation of why the specific transition is not allowed*

 *"""*

**def**\_\_init\_\_(self, previous, next, message):

self.previous = previous

self.next =next

self.message = message

Kebanyakan exception dapat diartikan dengan nama yang berakhir pada kata “error” mirip dengan penamaan exception standar

Banyak modul standar yang mengartikan exception mereka sendiri untuk melakukan pelaporan error yang mungkin terjadi pada function yang mereka definisikan.

1. **Defining Clean-up Actions**

Try statement memiliki opsi lain dimana digunakan untuk mendefinisikan sebuah pembersihan aksi yang harus di eksekusi pada seluruh keadaan Contoh :

**>>>try**:

**... raise**KeyboardInterrupt

**... finally**:

**...** print('Goodbye, world!')

**...**

Goodbye, world!

**KeyboardInterrupt**

Traceback (most recent call last):

 File "<stdin>", line 2, in <module>

Sebuah finally clause selalu dieksekusi sebelum meninggalkan try statement, dimana sebuah exception sudah terjadi atau belum. Ketika exception telah terjadi pada try clause dan belum ditangani oleh sebuah except clause, maka itu akan kembali bertambah setelah finally clause dijalankan. Finally clause juga dijalankan saat perjalanan keluar dimana semua clause lain dari try statemet ditinggal menggunakan break,continue atau return statement. Contoh Rumit :

**>>>def**divide(x, y):

**... try**:

**...**  result = x / y

**... except**ZeroDivisionError:

**...** print("division by zero!")

**... else**:

**...** print("result is", result)

**... finally**:

**...** print("executing finally clause")

**...**

**>>>**divide(2, 1)

result is 2.0

executing finally clause

**>>>**divide(2, 0)

division by zero!

executing finally clause

**>>>**divide("2", "1")

executing finally clause

Traceback (most recent call last):

 File "<stdin>", line 1, in <module>

 File "<stdin>", line 3, in divide

TypeError: unsupported operand type(s) for /: 'str' and 'str'

Seperti yang terlihat , finally clause dijalankan pada semua event. Type error bertambah dengan membagi 2 string yang tidak di tangani oleh except clause dan maka dari itu bertambah setelah finally clause dijalankan,

Seperti pada aplikasi di dunia nyata , finally clause sangat berguna untuk melepaskan sumber daya luar tanpa memperhatikan apakah sumber daya tersebut berhasil atau tidak.

1. **Predefined Clean-up Actions**

Beberapa objek didefinisikan sebagai standar clean up dari aksi yang diambil ketika objek sudah tidak lagi dibutuhkan, tanpa memperhatikan apakah operasi menggunakan objek tersebut berhasil atau tidak. Lihat contoh berikut

**for** line **in**open("myfile.txt"):

print(line, end="")

Masalah dengan kode ini adalah dimana meninggalkan file terbuka dalam waktu yang tidak dapat ditentukan. Setelah bagian ini kode selesai di jalankan. Ini bukan masalah dari script simple. Tapi dapat menjadi masalah untuk aplikasi yang lebih besar. With statement mengizinkan objek seperti file dapat digunakan untuk memastikan mereka selalu membersihkan secara benar dan tepat.

**with**open("myfile.txt") **as** f:

**for** line **in** f:

print(line, end="")

Setelah statement dijalankan , file akan selalu tertutup meskipun ada masalah yang dihadapi saat proses. Objek seperti file menyediakan sebuah aksi clean up yang akan memberikan indikasi di dalam dokumentasi mereka.

**BAB 11**

**LOGIC PROGRAMMING LANGUAGES**

1. **Penjelasan singkat mengenai Predicate Calculus**

Programming yang menggunakan bentuk logika simbolik sebagai Bahasa pemrograman sering disebut logika permrograman, dan Bahasa yang berdasar kepada logika simbolik disebut logika bahasa pemrograman atau Bahasa deklaratif.

Sebuah preposisi dapat dianggap sebagai pernyataan logika yang bisa benar atau salah. Preposisi mengandung objek dan hubungan antara objek-objek satu sama lainnya. Logika formal dikembangkan untuk menyediakan sebuah metode untuk mendeskripsikan preposisi, dengan tujuan unntuk mengizinkan preposisi yang dinyatakan secara formal untuk dilakukan pengecekan terhadap validitasnya.

Logika simbolik dapat digunakan untuk tiga dasar kebutuhan dari logika formal, yaitu: untuk mengekspresikan preposisi, umtuk mengekspresikan hubungan antara preposisi-preposisi, dan untuk mendeskripsikan bagaimana preposisi dapat disimpulkan dari preposisi lainnya yang diasumsikan bernilai benar. Bentuk khusus logika simbolik digunakan untuk pemrograman logika disebut predikat kalkulus.

1. **Preposisi**

Sebuah objek pada proposisi logika pemrograman yang diwakili oleh hal sederhana, yaitu konstanta atau variabel. Sebuah konstana adalah simbol yang mewakili sebuah Objek. Variabel adalah simbol yang dapat mewakili objek yang berbeda di waktu yang berbeda, meskipun dalam arti yang jauh lebih dekat dengan variabel dalam matematika daripada variabel dalam bahasa pemrograman imperatif. Proposisi sederhana, yang disebut proposisi atomik, terdiri atas compound term. Sebuah compound term merupakan salah satu unsur dari matematika relasi, ditulis dalam bentuk yang memiliki penampilan fungsi notasi matematika. Sebuah fungsi matematika adalah pemetaan, yang dapat direpresentasikan baik sebagai ekspresi atau sebagai meja atau daftar tupel.

Sebuah compound term terdiri dari dua bagian: sebuah functor, yang merupakan fungsi simbol yang memberi nama sebuah relasi, dan daftar parameter yang berurutan, yang bersama-sama mewakili unsur relasi. Sebuah compound term dengan parameter tunggal adalah 1-tupel; satu dengan dua parameter adalah 2-tupel, dan sebagainya.

Berikut merupakan contoh compound term:

* man(jake)
* like(bob, steak)

Proposisi dapat dinyatakan dalam dua bentuk:

* Fakta: proposisi yang dianggap benar
* Query: kebenaran proposisi akan ditentukan

Compound preposition:

* Memiliki dua atau lebih proposisi atomik
* Proposisi dihubungkan oleh operator

|  |
| --- |
| Logical Operators |
| Name | Symbol | Example | Meaning |
| Negation | ~ | ~a | Not a |
| Conjuction | ∩ | a ∩ b | a and b |
| Disjunction | ∪ | a ∪ b | a or b |
| Equivalence | ≡ | a ≡ b | a is equivalent to b |
| implication | ⊃⊂ | a ⊃ ba ⊂ b | a implies bb implies a |

|  |
| --- |
| Quantifiers |
| Name | Example | Meaning |
| Universal | ∀ x.p | Untuk semua x, p bernilai benar |
| existential | ∃ x.p | Terdapat sebuah x, dimana p bernilai benar |

1. **Overview of Logic Programming**

Bahasa yang digunakan untuk pemrograman logika disebut bahasa deklaratif, Karena program yang ditulis di dalamnya terdiri dari deklarasi daripada tugas dan laporan flow control. deklarasi ini sebenarnya pernyataan, atau proposisi, dalam logika simbolik. Salah satu karakteristik penting dari bahasa pemrograman logika adalah semantik, yang disebut semantik deklaratif. Konsep dasar semantik ini adalah bahwa ada cara sederhana untuk menentukan arti dari setiap pernyataan, dan tidak tergantung pada bagaimana pernyataan dapat digunakan untuk memecahkan masalah. Semantik deklaratif jauh lebih sederhana daripada semantik bahasa imperatif. Misalnya, makna proposisi yang diberikan dalam bahasa pemrograman logika dapat secara singkat ditentukan dari pernyataannya sendiri.

Dalam bahasa imperatif, semantik dari assignment statement sederhana membutuhkan pemeriksaan deklarasi lokal, pengetahuan tentang aturan scoping dari bahasa, dan bahkan mungkin pemeriksaan program di file lain hanya untuk menentukan jenis variabel dalam pernyataan penugasan. Kemudian, dengan asumsi ekspresi tugas berisi variabel, pelaksanaan program sebelum assignment statement harus ditelusuri untuk menentukan nilai-nilai dari variabel-variabelnya. Tindakan yang dihasilkan dari pernyataan itu, tergantung pada konteks run-time. Membandingkan semantik ini dengan proposisi dalam bahasa logika, tidak perlu mempertimbangkan konteks tekstual atau urutan eksekusi, jelas bahwa semantik deklaratif jauh lebih sederhana daripada semantik bahasa imperatif. Dengan demikian, semantik deklaratif sering dinyatakan sebagai salah satu keuntungan bahwa bahasa deklaratif memiliki nilai lebih daripada bahasa-bahasa imperatif.

1. **Terms**

Seperti dengan program dalam bahasa lain, program Prolog terdiri dari koleksi laporan. Hanya ada beberapa jenis laporan di Prolog, tapi mereka
dapat menjadi kompleks. Sebuah istilah Prolog adalah konstanta, variabel, atau struktur. Sebuah konstanta baik atom atau integer. Atom adalah nilai-nilai simbolik dari Prolog dan mirip untuk rekan-rekan mereka di LISP. Secara khusus, sebuah atom adalah baik string dari huruf, digit, dan garis bawah yang diawali dengan huruf kecil atau string apapun karakter ASCII dicetak dibatasi oleh apostrof.
Variabel adalah string dari huruf, angka, dan garis bawah yang diawali dengan
huruf besar atau garis bawah (\_). Variabel tidak terikat untuk jenis oleh
deklarasi. Pengikatan nilai, dan dengan demikian jenis, untuk variabel disebut
Instansiasi. Instansiasi hanya terjadi dalam proses penyelesaian. Sebuah variabel
yang belum ditetapkan nilai yang disebut uninstantiated. instantiations lalu
hanya selama diperlukan untuk memenuhi satu tujuan lengkap, yang melibatkan buktinya
atau pembantahan dari satu proposisi. variabel Prolog hanya kerabat jauh, di
baik dari segi semantik dan penggunaan, untuk variabel dalam bahasa-bahasa imperatif.
Jenis terakhir dari jangka disebut struktur. Struktur mewakili atom
proposisi predikat kalkulus, dan bentuk umum mereka adalah sama:
functor (daftar parameter) functor adalah setiap atom dan digunakan untuk mengidentifikasi struktur. Daftar parameter dapat berupa daftar atom, variabel, atau struktur lainnya. Seperti yang dibahas panjang lebar di subbagian berikut, struktur adalah sarana menentukan fakta-fakta di Prolog.
Mereka juga dapat dianggap sebagai obyek, dalam hal ini mereka memungkinkan fakta menjadi menyatakan dalam hal beberapa atom terkait. Dalam hal ini, struktur adalah hubungan, karena mereka menyatakan hubungan antara persyaratan. Struktur juga predikat saat konteksnya menentukan untuk menjadi query (pertanyaan.

1. **Fact Statements**

Laporan Prolog dimulai dengan pernyataan-pernyataan yang digunakan untuk membangun hipotesis, atau database diasumsikan informasi-laporan
dari mana informasi baru dapat disimpulkan. Prolog memiliki dua bentuk pernyataan dasar; ini sesuai dengan tanpa kepala dan klausa Horn menuju predikat kalkulus. Bentuk paling sederhana dari Horn tanpa kepala klausul dalam Prolog adalah struktur tunggal, yang diartikan sebagai tanpa syarat pernyataan, atau fakta. Logikanya, fakta hanya proposisi yang diasumsikan benar.

1. **Rule Statements**

Bentuk dasar lain dari pernyataan Prolog untuk membangun berkorespondensi Database ke Horn klausul menuju. Formulir ini dapat berhubungan dengan teorema dikenal di matematika dari mana kesimpulan dapat ditarik jika set kondisi tertentu puas. Sisi kanan adalah anteseden, dan sisi kiri adalah konsekuen. Jika yg dari pernyataan Prolog benar, maka konsekuen pernyataan juga harus benar. Karena mereka Horn klausa, konsekuen dari pernyataan Prolog adalah istilah tunggal, sementara anteseden bisa berupa istilah tunggal atau konjungsi. Konjungsi berisi beberapa istilah yang dipisahkan oleh logika AND operasi. Di Prolog, operasi AND tersirat. Struktur yang menentukan proposisi atomik dalam hubungannya dengan dipisahkan dengan koma, jadi satu bisa mempertimbangkan koma menjadi DAN operator.

1. **Goal Statements**

Sejauh ini, laporan Prolog menjelaskan tentang proposisi logis, yang
digunakan untuk menggambarkan kedua fakta yang diketahui dan aturan yang menjelaskan hubungan logis antara fakta. Pernyataan-pernyataan ini merupakan dasar untuk teorema-pembuktian model. teorema tersebut dalam bentuk proposisi yang kita ingin sistem baik membuktikan atau menyangkal. Dalam Prolog, proposisi ini disebut tujuan, atau query. Bentuk sintaksis laporan tujuan Prolog adalah identik dengan tanpa kepala klausa Horn yang sistem akan merespon ya atau tidak. Jawaban ya berarti bahwa sistem telah terbukti tujuannya adalah benar di bawah database yang diberikan fakta dan hubungan. Jawabannya tidak berarti bahwa baik tujuan bertekad untuk menjadi palsu atau sistem hanya dapat membuktikannya.

1. **The Inferencing Process of Prolog**

Bagian ini membahas resolusi Prolog. Efisien penggunaan Prolog mengharuskan programmer tahu persis apa sistem Prolog tidak dengan nya program. Query disebut tujuan. Ketika tujuan adalah proposisi majemuk, masing-masing fakta-fakta (struktur) disebut subgoal. Untuk membuktikan bahwa tujuan adalah benar, inferensia yang Proses harus menemukan rantai aturan dan / atau fakta inferensi dalam database yang menghubungkan tujuan untuk satu atau lebih fakta dalam database.

1. **Simple Arithmetic**

Prolog mendukung variabel integer dan aritmatika integer. Awalnya, aritmetik operator yang functors, sehingga jumlah 7 dan variabel X adalah
dibentuk dengan + (7, X). Prolog sekarang memungkinkan sintaks lebih singkat untuk aritmatika dengan operator. Operator ini membutuhkan ekspresi aritmatika sebagai operan kanan dan variabel sebagai operan kiri.

1. **List Structures**

Sejauh ini, satu-satunya struktur data Prolog yang telah kita bahas adalah proposisi atom, yang terlihat lebih seperti fungsi panggilan dari struktur data. Atom proposisi, yang juga disebut struktur, sebenarnya merupakan bentuk catatan. Struktur data dasar lain yang didukung adalah daftar. List adalah urutan apapun jumlah elemen, di mana unsur-unsur dapat atom, proposisi atomik, atau syarat-syarat, termasuk list lainnya. Prolog menggunakan sintaks dari ML dan Haskell untuk menentukan List. List elemen dipisahkan dengan koma, dan seluruh daftar dibatasi oleh kurung persegi.