**PROGRAMMING LANGUAGE CONCEPTS**

**(COMP6060-PLC)**



LAPORAN KONSEP BAHASA PEMROGRAMAN

(PYTHON)

Identitas Kelompok:

|  |  |  |
| --- | --- | --- |
| NIM | Nama | Posisi |
| 2001538761 | Geraldi Marcellino Taswin | Ketua |
| 2001535860 | Yohanzen Christanto Alexander | Anggota |
| 2001536573 | Felix Wenalvin | Anggota |
| 2001537241 | Alvin Ahasea Sianturi | Anggota |
| 2001591653 | Muhammad Fikrul Hadi Atsabit | Anggota |

|  |  |
| --- | --- |
| Kelas | LI01 |

DAFTAR ISI

[**BAB 1 INTRODUCTION** 5](#_Toc469592031)

[a) Alasan untuk belajar Concepts of Programming Languages 5](#_Toc469592032)

[b) Penjurusan programming 5](#_Toc469592033)

[c) Language Evaluation Criteria 5](#_Toc469592034)

[d) Influences on Language Design 6](#_Toc469592035)

[e) Language Categories 7](#_Toc469592036)

[f) Implementation Methods 8](#_Toc469592037)

[**BAB 2 SYNTAX AND SEMANTICS** 11](#_Toc469592038)

[a) Syntax 11](#_Toc469592039)

[b) Semantics 12](#_Toc469592040)

[**BAB 3 NAMINGS, BINDINGS, AND SCOPES** 13](#_Toc469592041)

[a) Naming: 13](#_Toc469592042)

[b) Binding 13](#_Toc469592043)

[c) Scope 13](#_Toc469592044)

[**BAB 4 DATA TYPES** 15](#_Toc469592045)

[a) Primitive Data Types 15](#_Toc469592046)

[b) Character String Types 15](#_Toc469592047)

[c) Enumeration 17](#_Toc469592048)

[d) Subrange Data Type 18](#_Toc469592049)

[e) Array 18](#_Toc469592050)

[f) Associative Array 18](#_Toc469592051)

[g) Records Data Types 19](#_Toc469592052)

[h) Tuple Types 19](#_Toc469592053)

[i) List Type 19](#_Toc469592054)

[j) Union Data Type 19](#_Toc469592055)

[k) Pointer 19](#_Toc469592056)

[**BAB 5 EXPRESSION AND ASSIGNMENT STATEMENTS** 20](#_Toc469592057)

[a) Expression 20](#_Toc469592058)

[b) Assignment Statement 22](#_Toc469592059)

[**BAB 6 CONTROL STRUCTURES** 25](#_Toc469592060)

[a) Selection Statements 25](#_Toc469592061)

[b) Iterative Statements 27](#_Toc469592062)

[**BAB 7 SUBPROGRAMS** 31](#_Toc469592063)

[**BAB 8 ABSTRACT DATA TYPES** 34](#_Toc469592064)

[a) The Concept of Abstraction 34](#_Toc469592065)

[b) Introduction to Data Abstraction 34](#_Toc469592066)

[c) Advantage of Data Abstraction 34](#_Toc469592067)

[d) Stack 35](#_Toc469592068)

[e) Queue 35](#_Toc469592069)

[f) Encapsulaption Constructs 36](#_Toc469592070)

[g) Naming Encapsulations 38](#_Toc469592071)

[**BAB 9 OBJECT-ORIENTED PROGRAMMING** 39](#_Toc469592072)

[a) Abstract Data Type 39](#_Toc469592073)

[b) Inheritence 39](#_Toc469592074)

[c) Polymorphism 40](#_Toc469592075)

[**BAB 10 CONCURRENCY** 42](#_Toc469592076)

[a) Thread 42](#_Toc469592077)

[b) Proses 42](#_Toc469592078)

[c) Perbedaan thread dengan proses 43](#_Toc469592079)

[**BAB 11 EXCEPTION AND EVENT HANDLING** 44](#_Toc469592080)

[a) Exception Handling 44](#_Toc469592081)

[b) Event Handling 47](#_Toc469592082)

[**BAB 12 FUNCTIONAL PROGRAMMING LANGUAGE** 49](#_Toc469592083)

[a) Ciri-ciri bahasa pemrograman fungsional: 49](#_Toc469592084)

[b) Keuntungan Functional Programming: 49](#_Toc469592085)

[c) Kekurangan Functional Programming: 49](#_Toc469592086)

[d) Pure Function 49](#_Toc469592087)

[e) Side Effect 50](#_Toc469592088)

[f) Using Functions 50](#_Toc469592089)

[g) Lambda 52](#_Toc469592090)

[**BAB 13 LOGIC PROGRAMMING LANGUAGE** 53](#_Toc469592091)

[**DAFTAR PUSTAKA** 56](#_Toc469592092)

[**LINK BLOG ANGGOTA KELOMPOK 1 (PYTHON)** 58](#_Toc469592093)

# BAB 1 INTRODUCTION

1. Alasan untuk belajar Concepts of Programming Languages

* Meningkatkan kemampuan mengekspresikan ide
* Meningkatkan kemampuan untuk memilih bahasa yang tepat
* Meningkatkan kemampuan untuk mempelajarin bahasa yang baru
* Pemahaman yang lebih baik tentang pentingnya penerapan
* Penggunaan yang lebih baik terhadap bahasa yang sudah dipahami sebelumnya
* Pemanfaatan keseluruhan dari komputasi

1. Penjurusan programming

* Scientific applications  
  Mendalami dalam teori komputasi dan desain sistem komputasi.
* Business applications  
  Mengembangkan applikasi untuk mempermudah/membantu dalam bidang business
* Artificial intelligence  
  Mengembangkan sistem yang memiliki kecerdasan manusia
* Systems programming  
  Mendalami algorithma dan penguasaan terhadap satu ato lebih bahasa pemrorgaman
* Web software  
  Membangun suatu website, dan mengelola isi website tsb

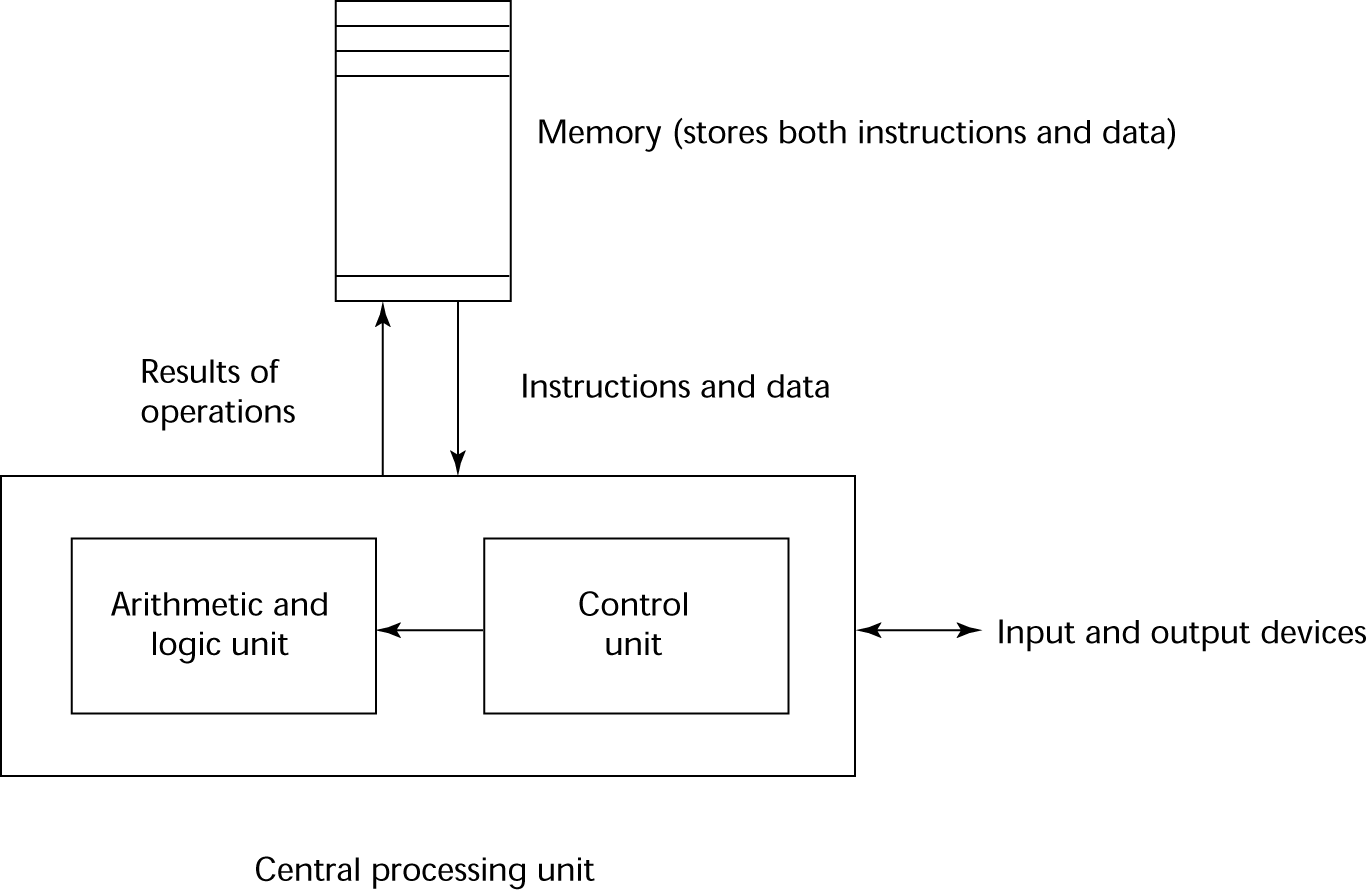
1. Language Evaluation Criteria

* Readability  
  Kemudahan dalam pembacaan suatu program karena syntax yang berlaku membuat   
  algoritma suatu program jadi lebih terstruktur
* Writeability  
  Kemudahan dalam penulisan suatu program dan mengakibatkan kesulitan membaca   
  bertambah karena kemudahan menulis adalah sedikit mengetik dan banyak singkatan
* Reliability  
  Konsep dari kebenaran suatu program atau verifikasi program
* Cost  
  Biaya akhir dari sebuah bahasa pemrograman

1. Cost of training programmers
2. Cost of writing programs
3. Cost of compiling programs
4. Cost of executing programs
5. Cost of Language implementation System
6. Cost of poor reliability
7. Cost of maintaining programs
8. Influences on Language Design

* Computer Architecture

Bahasa dikembangkan disekitar arsitektur komputer umum, yang dikenal sebagai arsitektur Von Neumann



* Program Design Methodologies  
  Metodologi pengembangan software baru (Misal: object oriented programming) menyebabkan paradigma pemrograman baru dan bahasa pemrograman baru
* Pada 1950-an dan awal 1960-an: Aplikasi sederhana, khawatir tentang standar mesin
* Pada akhir 1960-an: Standar orang menjadi penting, mudah dibaca, struktur kontrol yang lebih baik
* Pada akhir 1970-an: Process-oriented menjadi data-oriented (Data abstraction)
* Pada pertengahan 1980-an: Ditemukan Object-oriented programming (Data abstraction, inheritance, polymorphism)

1. Language Categories

* Imperative

1. Fitur utama adalah variabel, pernyataan penugasan, dan iteration
2. Termasuk bahasa yang mendukung pemrograman berorientasi objek
3. Termasuk scripting languages
4. Termasuk visual languages
5. Contoh: C, Java, Perl, JavaScript, Visual BASIC .NET, C++

* Functional

Sarana utama membuat perhitungan adalah dengan menerapkan fungsi untuk parameter yang diberikan

Contoh: LISP, Scheme

* Logic

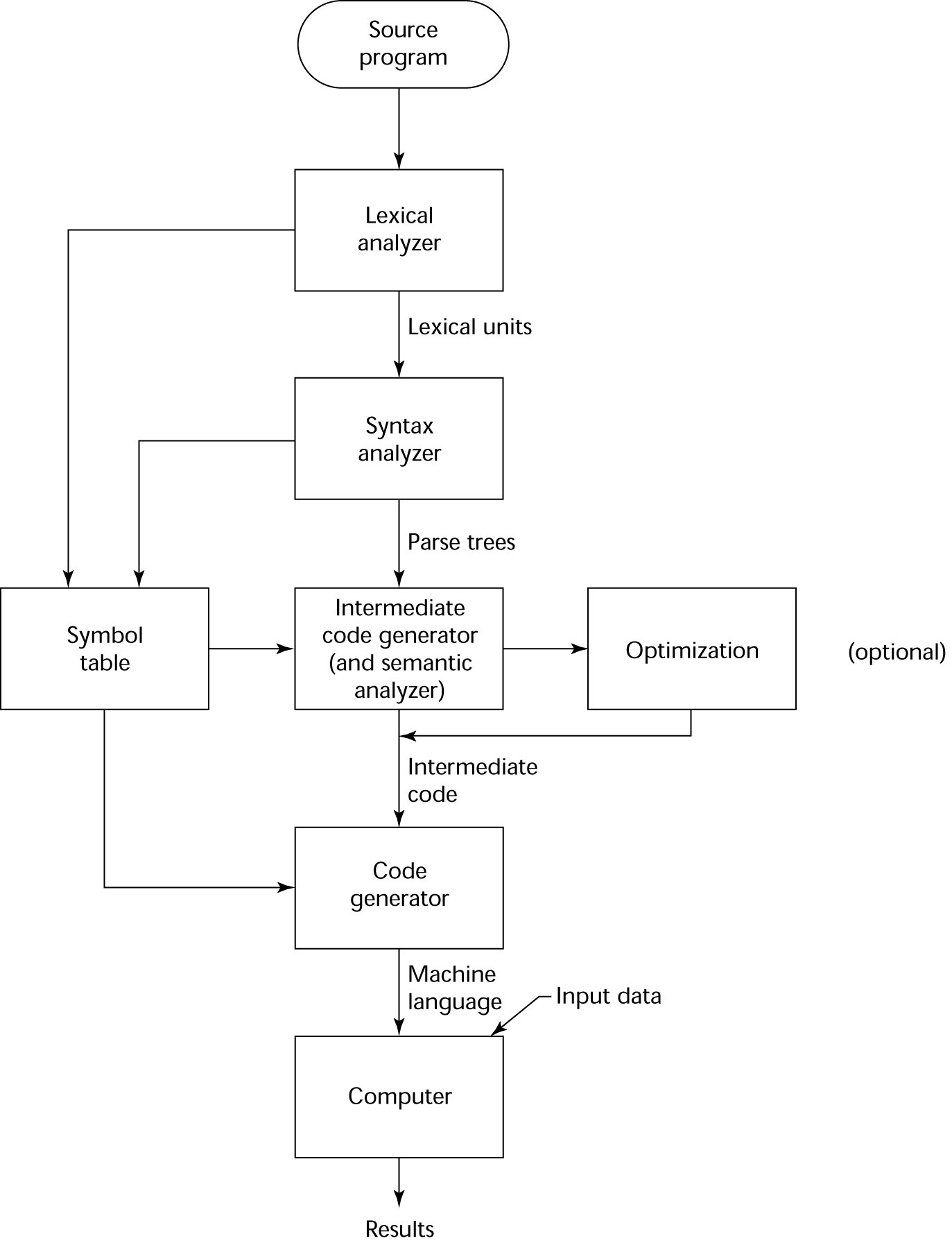
Berdasarkan aturan (aturan yang telah ditentukan)

Contoh: Prolog

* Markup/programming hybrid

Bahasa mark-up diperluas untuk mendukung beberapa program

Contoh: JSTL, XSLT



Lexical Analysis:

Mengubah karakter dalam program menjadi lexical units

Syntax Analysis:

Mengubah lexical units menjadi *parse trees* yang mewakilkan struktur sintaksis program

Semantics Analysis:

Menghasilkan kode intermediate

Code Generation:

Kode mesin dihasilkan

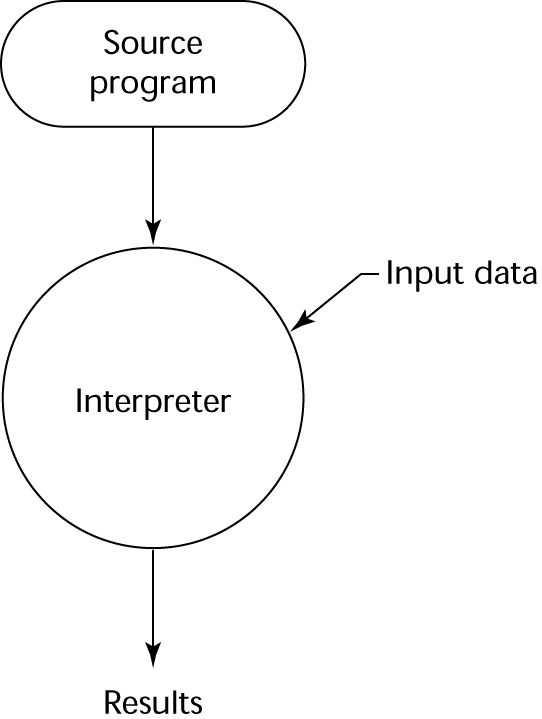
1. Implementation Methods

* Compilation

1. Program di terjemahkan menjadi bahasa mesin
2. Menggunakan aplikasi komersial besar
3. Penerjemahan lambat, eksekusi cepat
4. Phase dalam compilation

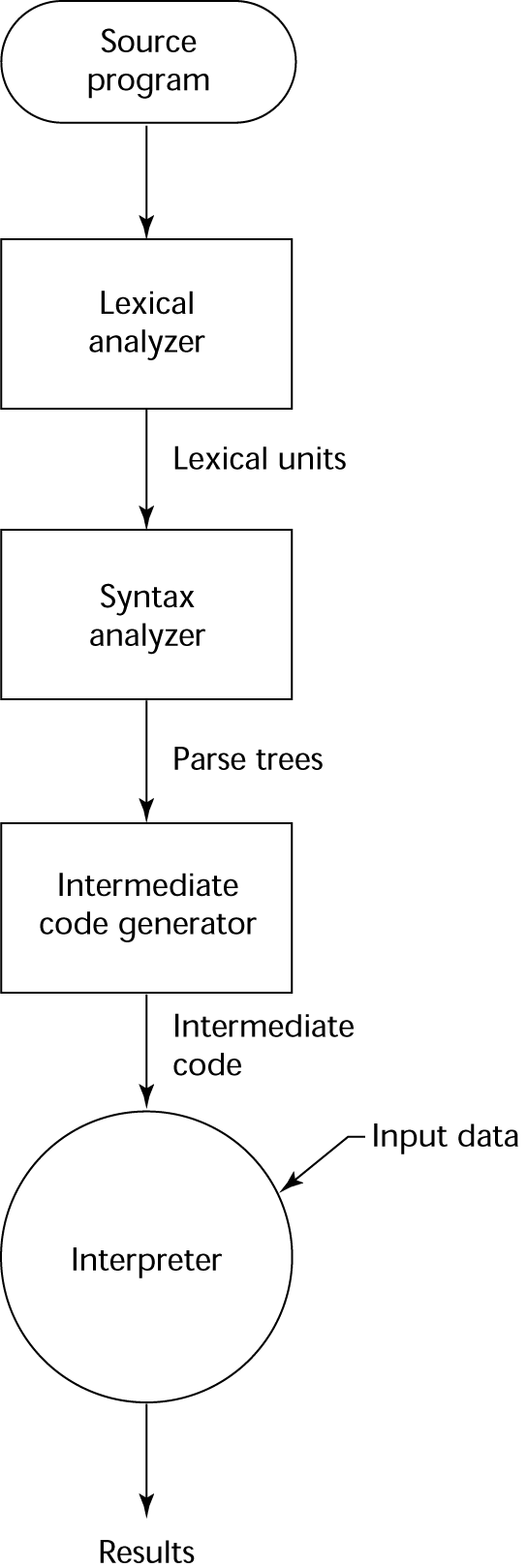
* Pure Interpretation

1. Program diinterpretasikan oleh program lain yang dikenal sebagai penerjemah
2. Menggunakan program kecil / ketika standar bukanlah suatu masalah
3. Implementasi program lebih mudah
4. Eksekusi lebih lambat (10 hingga 100 kali lebih lambat dari compiler)
5. Membutuhkan lebih banyak ruang



* Hybrid Implementation Systems

1. Gabungan antara compilation dan pure interpretation
2. Menggunakan program kecil dan sedang dimana standar bukan lah kondisi utama
3. Lebih cepat dari Pure Interpretation



# BAB 2

**SYNTAX AND SEMANTICS**

1. Syntax

* Syntax adalah sekumpulan aturan yang mendefinisikan bentuk atau pola suatu bahasa. Syntax mendefinisikan bagaimana suatu kalimat dibentuk sebagai barisan / urutan dari pemilihan suatu kata kunci yang dikonstruksi dengan karakter-karakter alphabet.
* Syntax mengatur cara mengkombinasikan kata-kata tersebut ke dalam suatu statement dengan bentuk yang benar dan mempermudah komunikasi antara programmer dengan pemroses bahasa pemrograman sehingga dapat disusun suatu program yang dapat berjalan dengan benar.
* Kriteria – kriteria syntax:
  1. Readability  
     Kemudahan dalam pembacaan suatu program karena syntax yang berlaku membuat algoritma suatu program jadi lebih terstruktur
  2. Writeability  
     Kemudahan dalam penulisan suatu program dan mengakibatkan kesulitan membaca bertambah karena kemudahan menulis adalah sedikit mengetik dan banyak singkatan
  3. Kemudahan Verifability  
     Konsep dari kebenaran suatu program atau verifikasi program
  4. Kemudahan Translasi  
     Suatu kebutuhan pemroses bahasa pemrograman. Kunci kemudahan translasi adalah keteraturan struktur suatu program, artinya suatu bahasa yang mempunyai syntax yang sulit dibaca dan ditulis oleh manusia, umumnya mudah untuk ditranslasikan
  5. Pengurangan ambiguitas  
     Ambiguitas merupakan masalah utama pada setiap perancangan suatu bahasa pemrograman. Suatu konstuksi ambiguitas membuat suatu statemen menjadi mempunyai dua arti atau lebih
* Beberapa elemen – elemen penting dari syntax:
  1. Himpunan karakter (ASCII)
  2. Identifier (Umumnya berupa string dan atau angka)
  3. Simbol atau Operator (Simbol matematika)
  4. Keyword (Kata – kata khusus yang mempunyai arti tersendiri)
  5. Komentar (# atau ‘’’ dalam bahasa python untuk comment)
  6. Delimiter (Awalan dan akhiran suatu ekspresi atau statement)

1. Semantics
   * Semantics adalah sebuah bahasa menggambarkan hubungan antara sintaks dan model komputasi. Dengan kata lain, semantics menjelaskan arti dari suatu program. Program yang benar dalam hal syntax belum tentu benar secara semantics.
   * Contohnya jika dengan syntax [subyek] + [predikat] + [objek]. Kita dapat membuat suatu kalimat seperti “Saya makan batu”. Secara syntax, kalimat itu adalah benar. Namun secara semantics, kalimat tersebut memiliki arti yang tidak benar.
   * Beberapa potensi semantics dalam bahasa pemrograman:
     1. Standarisasi Bahasa Pemrograman  
        Banyak usaha yang dilakukan untuk menstandarisasi bahasa pemrograman agar programmer dapat dengan lebih mudah menggunakannya
     2. Referensi untuk user  
        Programmer membutuhkan suatu dokumentasi yang pasti, supaya user dapat mengoperasikan program yang dibuat dengan baik
     3. Pembuktian dari pemrograman yang benar  
        Secara teori, program tidak mungkin bekerja dan berjalan jika tidak ada semantic
     4. Referensi untuk implementor  
        Semantic akan mencegah suatu gaya bahasa yang tidak kompatibel yang diwujudkan dalam suatu implementasi berbeda walaupun dengan bahasa yang sama
     5. Implementasi otomatis  
        Suatu tool dapat secara otomatis membuat translasi bahasa yang melebihi parsing. Hal ini dapat dilakukan jika semantic sudah dirumuskan
     6. Pemahaman yang lebih baik dari desain bahasa  
        Jika suatu rumusan semantic sulit untuk dideskripsikan secara formal maka rumusan semantic tersebut juga akan sulit digunakan oleh programmer. Sehingga sulit bagi programmer untuk menggunakan dan juga mengembangkannya

# BAB 3

**NAMING, BINDING AND SCOPES**

1. Naming:

Aturan Naming pada python hampirsamadengan C/C++ padadasarnya:

* Diawali dengan huruf atau underscore(\_)

Contoh:

1. Kuer (Valid)
2. \_kuer (Valid)
3. 8bit (Tidak valid karena diawali dengan angka)

* Penamaan menggunakan huruf dan underscore, juga boleh menggunakan angka selama tidak menjadi awal dari nama

Contoh:

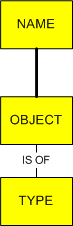
* num1, num, n\_um
* Perhatikan huruf besar/kecil.

Contoh:

* **Num** dan **num** berbeda karena kapitalisasi pada N

Usahakan menghindari penggunaan huruf besar i, huruf kecil L dan huruf O dengan 0 karena mereka mirip, akan lebih sulit membacanya.

1. Binding

Binding di python berjenis dynamic type bindings, yang artinya variabel yang digunakan tidak perlu diberi type (Ex: int, str, dll) untuk digunakan. Berbeda dengan static yang harus di declare terlebih dahulu untuk dapat digunakan

Susunan Binding secara dinamis dapat digambarkan seperti ini:

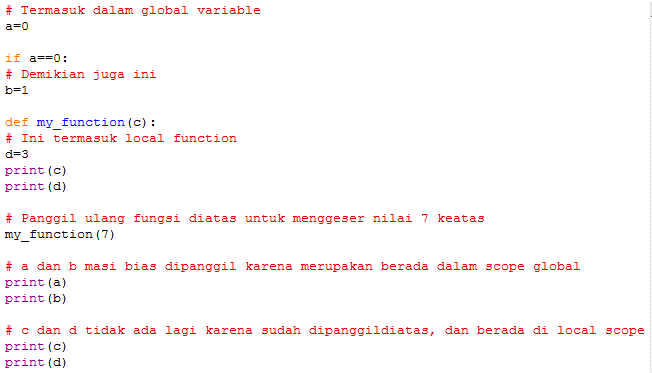
(Source: https://pythonconquerstheuniverse.wordpress.com/2009/10/03/static-vs-dynamic-typing-of-programming-languages/)

1. Scope

Scope adalah ruang lingkup yang dimana sebuah variabel dapat dibaca. Ada 2 jenis scope sesuai peletakkan variabelnya, yaitu Global Variable (yang dapat dibaca keseluruhan code) dan Local Variable (Yang hanya dapat dibaca dalam lingkup atau scope tertentu). Variabel yang di declare di atas file disebut “Global Variable”. Variabel tersebut dapat terus dipakai file tersebut atau oleh file lain yang menggunakan file tersebut.Hanya gunakan global variable untuk penggunaan fungsi untuk menghindari kesalahan yang tidak disengaja.

Lalu Variabel yang di declare di dalam suatu fungsi yang local dan hanya berlaku dalam fungsi itu disebut “Local Variable”. Variabel itu hanya berlaku dan dapat digunakan dalam scope fungsi tersebut. Diluar itu, variabel harus dideclare ulang agar dapat digunakan. Dalam local variable juga memuat suatu nilai yang dioper ke fungsi saat kita panggil. Saat penggunaan operator (=), operator akan secara otomatis membuat local variable, kecuali ada variable sama di scope yang sudah di declare.

Secara singkat tentang Scopes dalam code:



# BAB 4

**DATA TYPES**

*Data types* adalah koleksi sekumpulan data dan juga sekumpulan operasi yang masih belum didefinisikan kepada suatu objek.

*Descriptor* adalah koleksi dari berbagai atribut dari variable.

Suatu *object* mempresentasikan suatu tipe data yang abstrak.

1. Primitive Data Types

Hampir semua Bahasa pemograman menggunakan *Primitive Data Types. Primitive Data Types* merupakan suatu tipe data yang sangat umum digunakan pada semua Bahasa pemograman. Contohnya:

* Integer: Untuk angka
* Floating point: Untuk angka decimal
* Complex
* Decimal: Untuk angka decimal
* Boolean: Untuk menyatakan “TRUE” atau “FALSE”-nyasuatupernyataan
* Character untuk karakter.

Primitive Data Type pada Bahasa pemograman Python adalah:

* Boolean “TRUE” atau “FALSE”-nya suatu pernyataan
* Integer untukangka.
* Float untukangka decimal
* Str (String) untuksuatukumpulankarakter
* None untuktipe data kosong

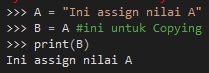
1. *Character String Types*

String merupakan kumpulan dari beberapa char atau karakter. Tipe data String untuk bebrapa Bahasa pemograman seperti java dan Python merupakan suatu tipe data primitive. Kecuali pada bahasa C. Pada Bahasa C, tidak ada tipe data String. Tetapi menggunakan primitive data type yaitu char. Pada Bahasa C mengunakan char “Array”.

Pada String Type Operations, ada typical operations, yaitu:

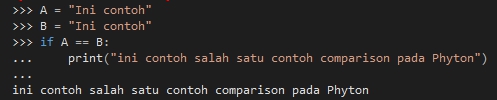
* Assingment and Copying

Contoh Assingment dan Copying pada Python

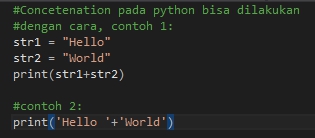


* Comparison (=, >, etc.)

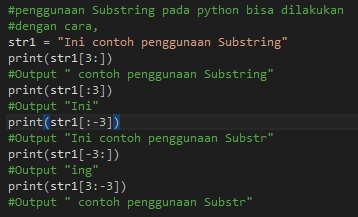
Contoh Comparison pada Python adalah:



* Concatenation

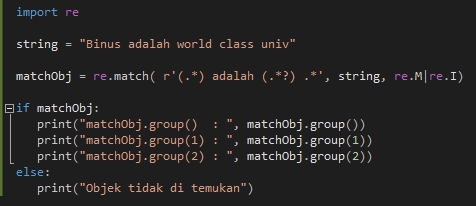


* Substring Reference

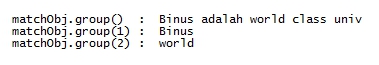


* Pattern Matching

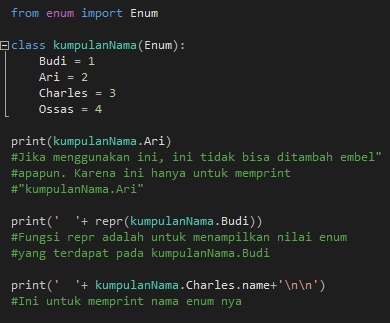
Pada Python, jika ingin menggunakn matching, harus meng-import “re” terlebih dahulu. Contohnya



Outputnya:



1. Enumeration

**

*Enumeration* atau enum baru diperkenalkan pada Python 3.4. Contoh penggunaan enum pada Python adalah:

Outputnya adalah:

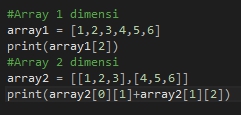
C:\Users\Fikrulhad Attsabit\AppData\Local\LINE\Cache\tmp\1481653050373.jpg

1. Subrange Data Type

*Subrange Data Type* adalah suatu jenis data yang memiliki rentang nilai hanya sebagian dari data utamanya. Sayangnya, pada Python tipe data ini tidak ada.

1. Array

*Array* merupakan suatu tipe data yang mencakup sekumpulan data yang dimasukkan oleh si programmer. Contohnya pada Python adalah:

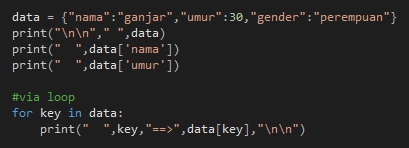


Outputnya:

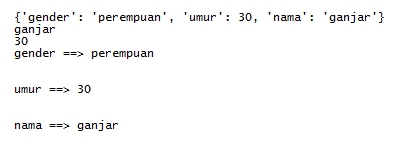
C:\Users\Fikrulhad Attsabit\AppData\Local\LINE\Cache\tmp\1481724252430.jpg

1. Associative Array

Merupakan, tingkat lanjut dari Array/List. Contohnya pada Python adalah:



Outputnya:



1. Records Data Types

*Records data type* pada Bahasa pemograman C dikenal dengan istilah “struct”. Tapi, pada Python, tidak ada istilah “Struct”. Untuk membuat struct pada Python diperlukan struct pada C terlebih dahulu, kemudian di import pada Python dengan mengimport “ctypes” terlebih dahulu.

1. Tuple Types

*Tuple type* pada Python hampir mirip dengan List atau Array. Cuma, pada Tuple, valuenya tidak dapat di ubah atau di update. Kemudian, cara mendeklarasikannya cukup dengan “iniTupple1 = (“asd”, “fgh”, 123, 456)”. Dan cara untuk printnya sama dengan cara memprint biasa.

1. List Type

*List data type* pada Python sama dengan Array. Mohon dilihat pada point 5.

1. Union Data Type

Sama halnya pada “struct” yang sudah di jelaskan pada poin ke-7. Perbedaanya adalah terletak pada datanya saja. Jika struct datanya bisa di input ke semua field, tetapi pada union, data tersimpan hanya pada field terakhir atau data terakhir yang diinput pada field.

1. Pointer

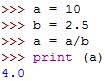
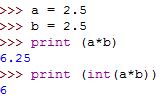
*Pointer* sering digunakan untuk menunjuk suatu variable yang kita buat. Pada Bahasa C biasanya menggunakan tanda “\*” (bintang) untuk menggambarkan bahwa variabel tersebut adalah pointer. Pointer pada Python tidak ada secara tersirat. Tetapi, algoritmanya ada. Seperti yang kita ketahui, penggunaan pointer biasa kita gunakan pada Bahasa C. danpada Python, bisa dikatakan jarang digunakan orang.

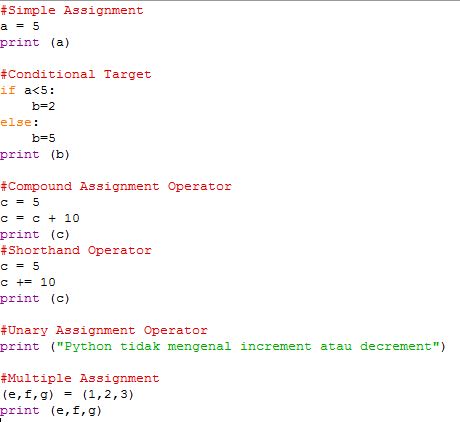
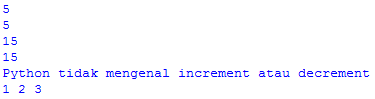
# BAB 5

**EXPRESSION AND ASSIGNMENT STATEMENTS**

1. Expression
   * Expression adalah dasar dari komputasi dalam menentukan perhitungan dalam sebuah bahasa pemrograman. Untuk memahami evaluation expression penting untuk mengenal urutan dari operator dan operand evaluation. Urutan dari evaluation operator diketahui dari associativity dan hukum dari bahasa pemrograman.
   * Arithmetic Expression  
     Arithmetic expression di dapat dari konvensi yang berevolusi dari matematika. Arithmetic expression terdiri dari operator(karakter khusus untuk memproses operand), operand(angka/karakter), parentheses(tanda kurung), dan function call. Operator itu sndiri bisa unary (hanya 1 operand), binary (ada 2 operand), atau ternary (ada 3 operand).
   * Operator Evaluation Order  
     Penentuan urutan dari evaluasi proses operator. Ada 3 jenis, yaitu:  
     1. Precedence  
        Precedence menggunakan nilai dari sebuah expression bergantung pada bagian yang di evaluasi dari operator pada sebuah expression. Karena pada beberapa kasus terjadi bahwa jika proses dibaca dari kiri ke kanan dan kanan ke kiri akan berbeda hasil. Hal itu yang menyebabkan bahwa akhirnya ditetapkan urutan. Dimana exponent berada di puncak, diikuti perkalian dan pembagian, lalu yang terakhir penjumlahan dan pengurangan.  
        Ex : 1+2\*3, jika kiri ke kanan hasilnya adalah 9. Namun jika kanan ke kiri hasilnya adalah 7. Dan jika mengikuti precedence yang ada hasilnya adalah 7.
     2. Associativity  
        Associativity adalah proses dimana jika level nya (operator) sama, maka akan dilakukan perhitungan biasa dari kiri ke kananAssociativity biasanya diproses dari kiri ke kanan kecuali exponent operator (ketika diberikan) terkadang dari kanan ke kiri.  
        Ex : 1+2-2+1, maka prosesnya adalah 1 +2 lalu -2 dan +1.
     3. Parentheses  
        Programmer dapat mengubah urutan dan aturan asosiasi dengan menempaatkan tanda kurung pada expression. Bagian yang di tanda kurungkan lebih diutamakan di banding yang tidak menggunakan tanda kurung.  
        Ex : (1+2)\*3, maka prosesnya akan menjumlahkan 1+2 terlebih dahulu baru \*3.
   * Overloaded Operator  
     Overloaded Operator adalah penggunaan sebuah operator yang memiliki lebih dari satu tujuan. Sehingga mempunyai potensi untuk menimbulkan ambiguisitas. Dalam beberapa bahasa pemrograman tertentu overloaded operator sedikit berbahaya, termasuk C karena operator \* dapat berarti kali dan bisa juga berarti pointer.
   * Type Conversion  
     Ada 2 jenis type conversion. **Narrowing** type conversion yaitu type conversion yang mengakibatkan penyempitan pada tipe data. Misalnya pada kasus tertentu dari float ke integer, adalah salah satu contoh Narrowing type conversion. **Widening** type conversion yaitu type conversion yang mengakibatkan pelebaran pada type data. Misalnya pada kasus tertentu terjadi konversi dari integer ke float, merupakan salah satu contoh dari Widenin type conversion.

Ada 2 cara type conversion, yaitu :

* + 1. Coercion in Expression  
       Salah satu bentuk decision desain tentang arithmetic expression apakah operator boleh memiliki operand dari tipe yang berbeda. Pemroses harus melakukan konversi tipe operand implicit karena computer tidak memiliki binary operations yang mengambil operand dari tipe data yang berbeda.  
       Contoh pada Python :  
         
       Contoh diatas menunjukan *Widening type conversion* dengan cara coercion atau implisit. Dimana dari tipe data a yang awalnya integer dipaksa berubah menjadi float.
    2. Explicit type conversion  
       Explicit type conversion adalah konversi tipe data secara explicit atau secara sengaja. Dapat dilakukan dengan fungsi seperti *cast* pada beberapa bahasa pemrograman yang menyediakannya  
         
       Contoh pada Python :  
         
       Contoh diatas menunjukan *Narrowing type conversion* secara eksplisit. Dimana dari tipe data a dan b yang awalnya float, dilakukan cast menjadi integer dengan fungsi int().
  + Short-circuited evaluation  
    Short-circuited evaluation adalah expression dimana hasil dari salah satu nya ditentukan tidak melalui hasil evaluasi dari semua operand dan operator. Hanya untuk mempersingkat waktu complie.  
    Contoh pada Python :  
      
    Contoh diatas menunjukan ketika a = 0, lalu dikali angka berapapun hasilnya akan 0 juga. Maka dengan adanya Short-circuited evaluation, bagian dibelakang \* kedua tidak akan dihitung dan dibiarkan 0 karena angka berapapun dikali 0 akan 0 juga.

1. Assignment Statement
   * Assignment statement adalah bagian tengah dari susunan bahasa imperative. Dia memberikan mekanisme yang user dapat secara dinamis dapat mengubah binding dari nilai variable. Ada 5 jenis assignment statement, yaitu :
     1. Simple Assignment  
        Hampir semua bahasa pemrograman menggunakan tanda ‘=’ untuk melakukan simple assignment ini.
     2. Conditional Target  
        Assignment yang akan terjadi jika dan hanya jika suatu kondisi terpenuhi.
     3. Compound Assignment Operator  
        Metode assignment yang menggunakan binary operator dimana operand pertamanya merupakan variable atau dirinya sendiri. Namun biasanya hal ini sudah jarang digunakan karena programmer lebih suka menggunakan *shorthand operators*
     4. Unary Assignment Operator  
        Metode assignment yang menggunakan unary operator. Yaitu dengan cara menggabungkan operator increment atau decrement pada assignment. Operator ++ sebagai increment dan operator -- sebagai decrement, bisa digunakan sebagai expression atau bentuk berdiri sendiri. Dapat muncul sebagai prefix operator maupun sebagai postfix operator.
     5. Multiple Assignment  
        Metode assignment dimana target dan value saat melakukan assignment adalah berjumlah lebih dari satu. Dan juga bagi bahasa pemrograman yang menyediakan fungsi ini memberikan kemudahan pagi programmer untuk menukan value dari beberapa target atau variablenya.  
          
        Contoh Assignment Statements pada Python :  
          
        Outputnya :  
        

**BAB 6**

**CONTROL STRUCTURES**

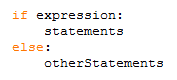
*Control Structure* adalah statement yang mengontrol statement-statement yang dijalankannya. *Control Structure* memuat Selection dan Iteration.

1. *Selection Statements* adalah statement yang mementukan statement apa yang akan dijalankan. Jumlah pilihan eksekusi bisa 2 atau lebih. Selection yang memiliki 2 kemungkinan eksekusi disebut *Two-Way Selection*, sementara yang memiliki >2 kemungkinan disebut *Multiple-Way Selection*.

Bentuk argumen dalam Python dapat berupa aritmatika dan boolean dengan penggunaan operator aritmatika dan boolean. Dalam Python, block statement dapat ditentukan dengan menggunakan indentasi. Maka semua statement yang terindentasi sama akan dianggap sebagai 1 block.

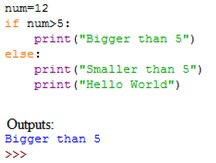
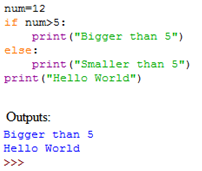
Two-Way Selection dalam Python

Dapat dilakukan dengan **if..else**, formatnya sebagai berikut:

**

Perhatikan indentasi dari statements dan otherStatements, indentasi tersebut menunjukkan bahwa mereka masing-masing bagian dari block **if** dan **else**. Dan dapat dicatat bahwa **else** tidak wajib ditambahkan, karena Python akan dengan otomatis menambahkan **else** dengan block statement kosong.

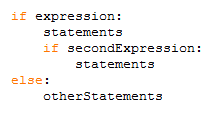
Contoh dari **if..else** Statement:



Perhatikan kedua contoh tersebut. Pada gambar di sebelah kiri, **print("Hello World")** dianggap berada diluar block **if..else**. Sementara pada gambar di sebelah kanan, **print("Hello World")** adalah bagian dari block statement dari **else**, yang tidak dijalankan karena kondisi **if** telah terpenuhi.

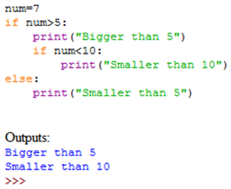
Untuk membuat sebuat Selection yang lebih spesifik, dapat menggunakan Nested **if**. Artinya, menaruh statement **if** didalam statement **if**.

Nested **if** memiliki format sebagai berikut:



Jumlah **if** yang dapat di*nest* tidak terbatas, jadi dapat ditambahkan sesuai keperluan.

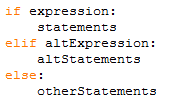
Contoh Nested **if**:



Multiple-Way Selection

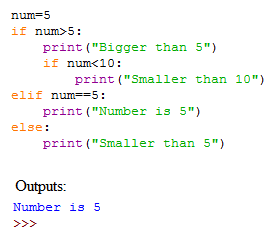
Multiple-Way Selection dalam Python dapat dilakukan dengan menggunakan **elif** atau yang sering disebut dengan "else if".

Format dari **elif** adalah sebagai berikut:



Cara kerjanya adalah ketika kondisi **if** tidak terpenuhi, program akan menghiraukan block statement milik **if** dan mengecek kondisi dari **elif**, jika kondisi **elif** terpenuhi, maka blocks statement **elif** akan dijalankan, selain itu, program akan terus mengulang pengecekan sampai ada **elif** yang terpenuhi atau sampai **else**.

Contoh Multiple-Way Selection menggunakan **elif**:



Perlu dicatat bahwa Python tidak memiliki fungsi **switch..case**, namun dapat digantikan dengan menggunakan **if..elif..else** atau *dictionary*, tetapi *dictionary* tidak akan dibahas untuk sekarang.

1. *Iterative* *Statements* adalah statement yang jika kondisinya terpenuhi, akan mengiterasi/mengulang block statementnya. Umumnya disebut *looping*.

Jenis-jenis Iterative Statements

* Logically-Controlled Loops

Loop yang dieksekusi selama kondisi tertentu terpenuhi, identik dengan **if.**

Ada 2 jenis Logically-Controlled Loops, yaitu:

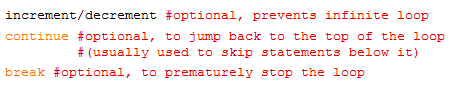
1. Pre-Test Controlled Loops (while)
2. Post-Test Controlled Loops (do..while)

Dalam bahasa pemrograman Python, hanya ada Pre-Test Controlled Loop yatu **while**. Mengimitasi do..while dalam Python dapat dilakukan namun tidak akan dibahas untuk sekarang.

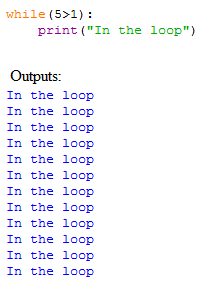
**while** dapat digunakan dengan format berikut:



Ada pula beberapa statement pelengkap **while** sebagai berikut:

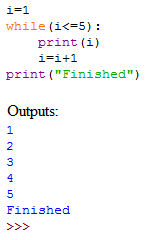
****

Contoh penggunaan **while** loop:



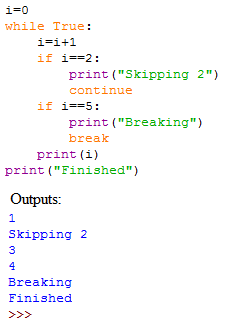
Selama kondisi **while** terpenuhi, **print("In the loop")** akan terus dijalankan berkali-kali. Namun, dengan ini, program akan terjebak didalam *infinite loop* dan tidak bisa keluar dari loop **while(5<1)**. Untuk menghindari *infinite loop*, maka digunakanlah increment serta break (jika diperlukan).

Contoh penggunaan **while** loop dengan increment:



Dapat dilihat bahwa ketika nilai i>5, kondisi **while** loop akan menjadi false. Maka loop dapat berhenti dan keluar dari block statement **while**.

Untuk memahami mengenai penggunaan **continue** dan **break,** perhatikan contoh loop berikut:



Program diatas akan menampilkan nilai **i**. Jika nilai **i=2**, maka program akan kembali ke awal loop, oleh sebab itu **print(i)** untuk **i=2** tidak dijalankan. Lalu perhatikan kondisi **while** loop yang bernilai **True**! Kondisi tersebut akan memulai *infinite loop* namun dapat berhenti ketika nilai **i=5** dengan bantuan **break**.

* Counter-Controlled Loops

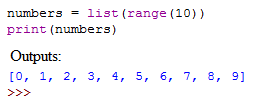
Counter-Controlled Loop adalah loop yang memanfaatkan tiga buah statement. "Counter", expression, dan increment.

Counter-Controlled Loop dalam bahasa pemrograman Python adalah loop yang disebut **for**, namun logika dari formatnya berupa for..each, berbeda dari for loop yang umum ditemukan pada Java/C++.

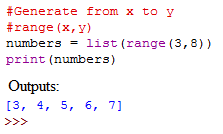
Sebelum mempelajari format loop **for**, ada object yang disebut **range()**. **range()** adalah object generator angka yang menyediakan sebuah list angka sebanyak yang diminta, dimulai dari nol.

Agar lebih mengerti cara kerja **range()**, perhatikan contoh berikut:

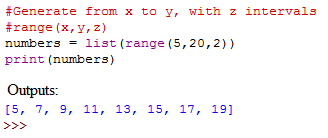
Untuk generasi angka dari nol sampai x, gunakan **range(x)**:



Untuk memulai dari nilai tertentu, gunakan **range(x,y)**:



Untuk lebih spesifik lagi, gunakan 3 parameter, **range(x,y,z)**:



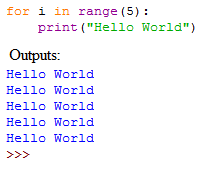
Setelah memahami konsep penggunaan **range()**, maka mari lanjut mengenai **for** loop!

Berikut adalah format untuk penggunaan **for** loop:



**for** loop akan terus beritasi sampai **counter** bernilai sama dengan **range()** atau panjang **list** (tergantung yang dipakai yang mana), seperti namanya, Counter-Controlled Loop.

Contoh dari **for** loop untuk menampilkan "Hello World" sebanyak 5 kali:

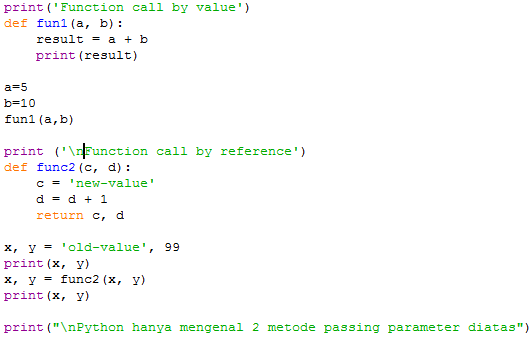
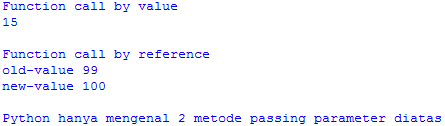


# BAB 7

**SUBPROGRAM**

Subprogram

* + Subprogram adalah pondasi block dari sebuah program yang memiliki bentuk terpentitng dalam konsep design bahasa pemrograman.
  + Dasar dari setiap subprogram adalah:
    1. Memiliki sebuah subprogram pada setiap single entry point
    2. Pemanggilannya tergantung selama eksekusi dari subprogram yang dipanggil
    3. Memberikan suatu nilai balik kepada pemanggil subprogram yang dipanggil
  + Metode dalam passing parameter di subprogram:
    1. Pass by value  
       Mengoper parameter yang dibutuhkan subprogram dengan hanya mengoper nilai dari suatu variable
    2. Pass by result  
       Mengoper parameter yang dibutuhkan subprogram dengan mengoper kembali ke pemanggil subprogram soal hasil dari proses subprogram tersebut
    3. Pass by value-result  
       Mengoper parameter yang dibutukan subprogram dengan nilai dari suatu variable lalu si pemanggil subprogram akan menerima kembali hasil dari proses subprogram tersebut
    4. Pass by reference  
       Mengoper parameter yang dibutuhkan subprogram dengan berupa alamat yang nantinya isi dari alamat tersebut akan digunakan lalu akan diganti sesuai dengan proses subprogram tersebut
    5. Pass by name  
       Hampir sama dengan pass by reference namun pada metode ini yang dioper bukan alamat namun nama sebuah variable. Metode ini hanya dapat ditemukan di beberapa bahasa pemrograman

Contoh subprogram di Python :  
  
Outputnya :  


* + Ada 3 jenis binding yang biasa ditemukan di subprogram:
    1. Shallow binding  
       Variable atau value yang digunakan dan terpengaruh adalah yang disekitar pemanggil subprogram tersebut
    2. Deep binding  
       Variable atau value yang digunakan dan terpengaruh hanya yang ada didalam subprogram tersebut. Sehingga tidak mengubah yang diluar subprogram
    3. Ad Hoc Binding  
       Variable atau value yang digunakan hanya yang dioper atau di passing kedalam subprogram tersebut dan tidak akan mempengaruhi apapun. Karena hanya bersifat menerima.
  + Overloaded Subprogram  
    Terjadi ketika ada 2 atau lebih subprogram dengan nama yang sama. Sehingga menciptakan ambiguitas
  + Closures  
    Pengakhiran atau penghentian dari sebuah subprogram. Dibutuhkan ketika :
    1. Subprogram dapat dipanggil oleh dirinya sendiri (recursion)
    2. Subprogram mengakses variable pada nesting scopes dan dapat dipanggil dari mana saja.
  + Coroutines  
    Coroutines adalah ketika sebuah subprogram dijalankan secara berulang. Ciri utama dari coroutines adalah memiliki multiple entry dan pengontrol untuk berhenti juga ada pada dirinya sendiri.

# BAB 8 ABSTRACT DATA TYPES

1. The Concept of Abstraction  
   Abstraction adalah pandangan ato representasi dari sebuah entitas yang hanya mencakup atribut yang paling signifikan. Konsep Abstraction adalah fundamental dalam pemrograman (dan ilmu komputer). Hampir semua bahasa pemrograman mendukung proses Abstraction dengan subprogram. Hampir semua bahasa pemrograman yang dirancang sejak tahun 1980 mendukung data abstraction.
2. Introduction to Data Abstraction  
   Abstract data type adalah tipe data yang memenuhi dua kondisi berikut:

* Representasi tipe objek disembunyikan dari unit program yang menggunakan objek tersebut, sehingga satu-satunya operasi yang mungkin adalah yang disediakan dalam definisi type ini.
* Deklarasi type dan protokol operasi pada type objek disimpan dalam unit syntactic tunggal. Program unit lain diperbolehkan untuk membuat variabel type yang didefinisikan.

1. Advantage of Data Abstraction  
   Kondisi pertama:

* Reliability: Dengan menyembunyikan representasi data, user codwe tidak dapat langsung mengakses objek dari type atau tergantung dari representasi, memungkinkan representasi diubah tanpa mempengaruhi user code.
* Mengurangi berbagai kode dan variabel yang programmer harus sadari
* Konflik penamaan cenderung sedikit.

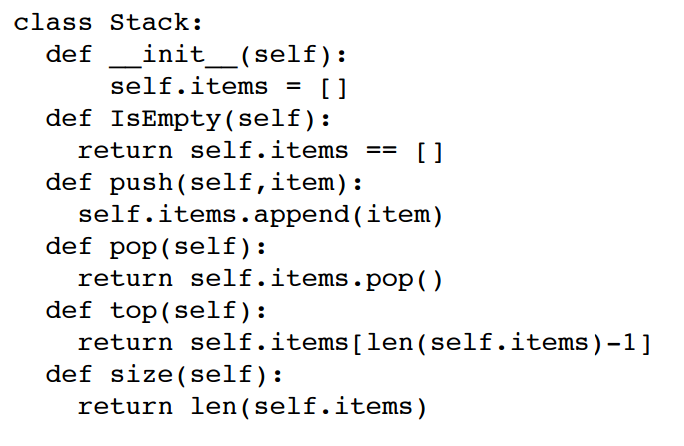
Kondisi kedua:

* Menyediakan metode organisasi program
* Mendukung modifikasi ( segala sesuatu yang berkaitan dengan struktur data )
* Kompilasi dilakukan secara terpisah

Dalam python ADT (Abstract Data Types) disebut Classes

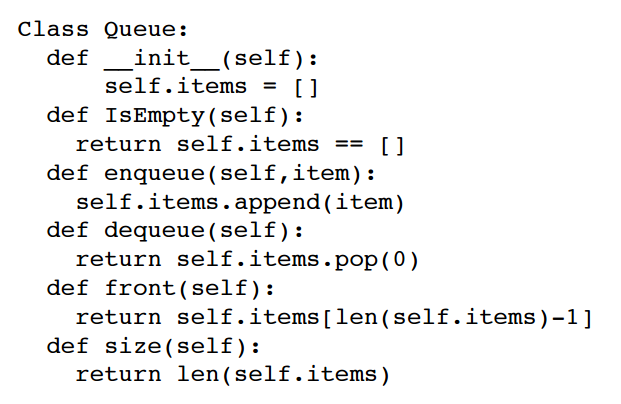
ADT merupakan gagasan yang sangat berguna dalam pemrograman bahasa  
  
Python memiliki 2 macam ADT:

1. Stack
2. Queue
3. Stack  
   Stack merupakan LIFO ( last in, first out) dengan operasi berikut

* Stack(): Membuat stack baru yang kosong. Perlu ada parameter dan return ke stack kosong
* Push(item): menambahkan item baru ke bagian stack paling atas. Perlu item dan return nothing
* Pop(): Menghapus item teratas dari stack. Perlu ada parameter dan return item. Stack dimodifikasi
* Top(): Mengembalikan item teratas dari stack tetapi tidak menghapusnya. Tidak butuh parameter. Stack tidak dimodifikasi
* IsEmpty(): Melakukan tes untuk melihat apakah stack kosong. Perlu ada parameter dan return boolean value
* Size(): Mengembalikan jumlah item pada stack. Tidak butuh parameter dan return integer  
    
  Contoh Stack:  
  

1. Queue  
   Queue merupakan FIFO ( first in, first out ) dengan operasi berikut

* Queue(): Membuat queue baru yang kosong. Tidak perlu parameter dan return queue kosong
* Enqueue(item): Menambahkan item baru ke bagian belakang queue. Perlu item dan retun nothing
* Dequeue(): Menghapus item dari depan queue. Perlu ada parameter dan return item. Queue dimodifikasi
* Front(): Mengembalikan item depan dari queue tetapi tidak menghapusnya. Tidak perlu parameter. Queue tidak dimodifikasi
* isEmpty(): Melakukan tes untuk melihat apakah queue kosong. Perlu ada parameter dan return boolean
* size(): Mengembalikan jumlah item pada queue. Perlu ada parameter dan return integer.

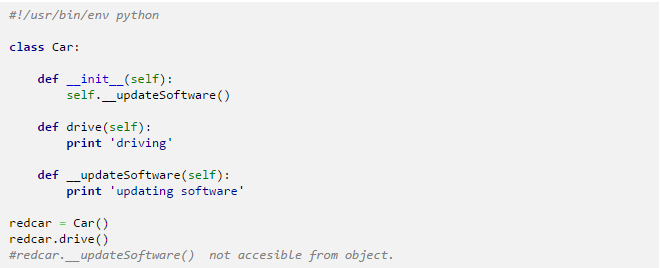
Contoh queue:  


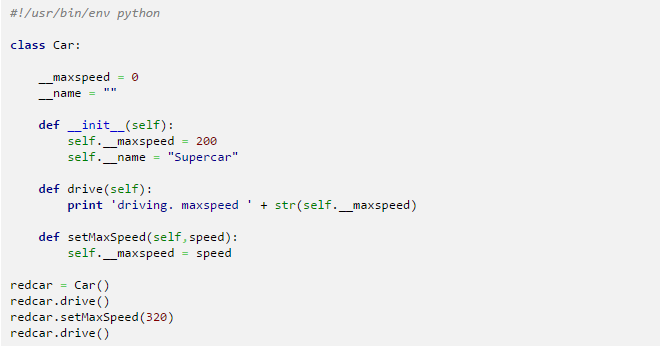
1. Encapsulaption Constructs  
   Program besar butuh dua kebutuhan khusus:

* Beberapa caraorganisasi, selain hanya pembagian ke dalam subprogram
* Beberapa cara kompilasi parsial (unit kompilasi yang lebih kecil dari seluruh program)

Pengelompokkan subprogram yang berhubungan secara logis menjadi unit dapat dikompilasi secara terpisah (disebut encapsulation)  
  
Python dapat mengorganisir program dengan definisi subprogram didalam logical subprogram yang lebih besar yang menggunakannya.  
  
Ada 4 tipe:

1. Public method  
   Dapat diakses darimana saja
2. Private method  
   Hanya dapat diakses dalam kelas mereka sendiri. Dimulai dengan dua garis bawah ( \_\_ )
3. Public variable  
   Dapat diakses darimana saja
4. Private variable  
   Hanya dapat diakses dalam kelas mereka sendiri atau dengan metode jika didefinisikan. Dimulai dengan dua garis bawah ( \_\_ )

Contoh Private method  
  
  
Akan menghasilkan   


Contoh Private variables  


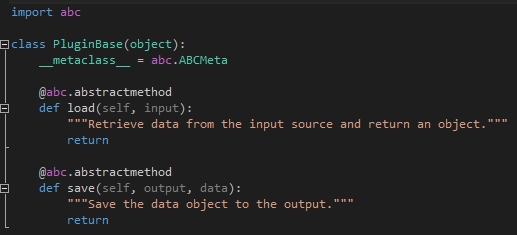
1. Naming Encapsulations  
   Program besar menentukan banyak nama global, dibutuhkan cara untuk membaginya menjadi logical groupings. Naming Encapsulation digunakan untuk membuat scope baru untuk penamaan.

**BAB 9**

**OBJECT-ORIENTED PROGRAMMING**

1. Abstract Data Type

Pengertian Abstract Data Type menurut Wikipedia adalah: model matermatika yang merujuk pada sejumlah bentuk struktur data yang memiliki kegunaan atau perilaku yang serupa; atau suatu tipe data dari suatu bahasa pemrograman yang memiliki sematik yang serupa. Pada Python, penggunaan Abstract Data Type (ADT) dapat dilakukan dengan cara meng-import “abc”. Contohnya pada Bahasa Python adalah:

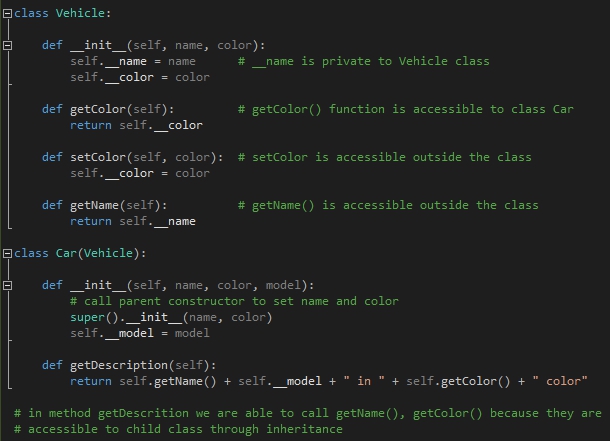


(Sumber contoh: <https://pymotw.com/2/abc/>)

Dikarenakan ini hanya masih mendeklarasikan datanya, dan belum digunakan, maka outputnya tidak ada.

1. Inheritence

Inheritance atau pewarisan pada pemrograman berorientasi objek merupakan suatu hubungan dua buah kelas atau lebih. Dalam hal ini ada kelas yang memiliki atribut dan metode yang sama dengan kelas lainnya beserta atribut dan metode tambahan yang merupakan sifat khusus kelas yang menjadi turunannya.Contohnya pada Python adalah:



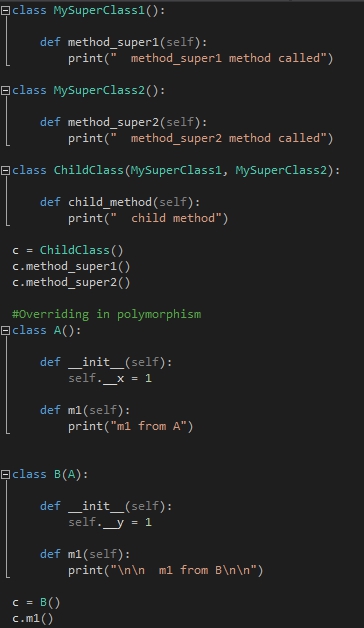
(Sumber gambar: <http://thepythonguru.com/python-inheritance-and-polymorphism/>)

Outputnya:

C:\Users\Fikrulhad Attsabit\AppData\Local\LINE\Cache\tmp\1481730483595.jpg

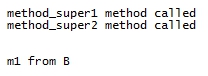
1. Polymorphism

*Polymorphism* dapat berarti banyak bentuk, maksudnya yaitu kita dapat menimpa (override), suatu method, yang berasal dari parentclass (super class) dimanaobject tersebut diturunkan, sehingga memiliki kelakuan yang berbeda. Contoh Polymorphism pada Python adalah:



(Sumber contoh: <http://thepythonguru.com/python-inheritance-and-polymorphism/>)

Outputnya:



# BAB 10

**CONCURRENCY**

Teknik dasar pada concurrency di python adalah greenlets, pthreads(threading untuk python) dan process (multiprocess python). Saat menggunakan thread, kita akan mencapai concurrency atau parallelism. Perbedaan keduanya adalah fokusnya. Concurrency lebih kepemerataan beberapa thread, perallelisme menggunakan beberapa prosesor untuk memperlancar jalannya program.

1. Thread

Thread adalah komponen dasar jalannya CPU. Bisa juga disebut “proses ringan”. Thread adalah urutan instruksi didalam proses dan beraksi seperti adanya proses didalam proses, tetapi berbeda dari proses karena dia tidak punya kumpulan informasi tentang proses itu sendiri. Biasanya, beberapa thread diciptakan dalam proses. Thread dijalankan dengan proses dan beberapa proses dijalankan dalam lingkup os tersebut.

Thread terdiri dari:

1. Thread ID
2. Counter Program
3. Register Set
4. Stack

Thread juga membagi sumbernya dengan thread lain yang terdiri dari:

1. Code Section
2. Data Section
3. Semua operator yang tersedia

Dalam multithread, satu server thread mungkin akan diblok untuk menunggu sesuatu, dan thread lain pada tempat yang sama tetap berjalan. Jika proses terhalang, maka seluruh proses tersebut berhenti. Tapi multithread process berguna dalam program seperti web browser saat men-download file, melihat animasi disaat bersamaan.

1. Proses

Dalam komputer, proses adalah bagian program komputer yang sedang dijalankan. Proses terdiri dari kode program itu dan apa yang sedang dijalankan. Banyaknya proses yang terdiri dari thread untuk menjalankan perintah akan bervariasi tergantung operating system (OS) masing – masing komputer. Kebanyakan OS modern mencegah komunikasi langsungantara proses mandiri, memberikan inter-process communication (IPC) yang sangat teratur dan terkontrol.

Proses terdiri dari beberapa hal:

1. Gambaran kode yang berhubungan dengan program
2. Memory
3. OS yang menyediakan tempat untuk memproses data.
4. Atribut keamanan (permission)
5. Memori Fisik

OS menyimpan proses terpisah dan juga bahan yang mereka butuhkan, sehingga bahan tersebut berkemungkinan kecil untuk bertabrakan dan mengakibatkan kerusakan sistem.

1. Perbedaan thread dengan proses

Proses menjalankan perintah dari aplikasi. Saat kita menjalankan kode Python, kita memulai proses yang menjalankan script python itu. OS menyediakan proses yang menjawab peluncuran script tersebut dan thread utama proses itu mulai bekerja.

Thread adalah arah jalannya proses, dan dalam program python, dapat menghasilkan beberapa thread. Sebagai contoh, hal ini sudah di-implementasikan lewat pthreading.

Dalam sebuah processor, multithread sering terjadi dalam multitasking (contohnya saat prosesor itu berganti ke thread yang berbeda.)

Pergantian ini terjadi cukup cepat sehingga kita membuat thread berjalan di saat bersamaan. Dalam multiprocessor, thread menjadi concurrent, dengan setiap core CPU menjalankan thread terpisah secara bersamaan.

**BAB 11**

**EXCEPTION AND EVENT HANDLING**

1. Exception Handling

*Exception* adalah kejadian-kejadian yang tidak biasa, baik keliru maupun tidak, yang membutuhkan penanganan khusus. Contohnya ketika memasukan input char ketika program meminta integer. *Exception handler* adalah unit yang memproses exception-exception tersebut.

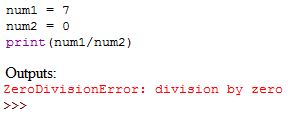
Bahasa-bahasa pemrograman yang tidak memiliki *Built-in Exception Handling*, masih dapat membuat exception handler sendiri. Namun code untuk mendeteksi error sangat rumit dan dapat mengotori program.

Python adalah salah satu bahasa pemrograman yang memiliki *Built-in Exception Handling*.

Jenis-jenis **exception** yang umum ditemukan dalam Python:

* ImportError: ketika sebuah *import* tidak dapat dilaksanakan
* IndexError: memanggil list dengan index yang *out-of-range*
* NameError: memanggil variable yang tidak diketahui
* SyntaxError: ketika terdapat kesalahan dalam syntax
* TypeError: sebuah function dipanggil menggunakan tipe yang tidak cocok
* ValueError: sebuah function dipanggile menggunakan tipe yang cocok, namun dengan nilai yang tidak sesuai

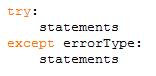
Contoh dari **exception**:



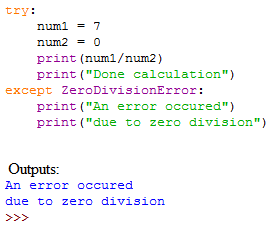
Cara untuk mengatasi **exception** seperti diatas agar program tidak diterminasi dapat dilakukan dengan **exception handler**.

**try..except:**

Berikut adalah format **exception handler** Python, **try..except**:

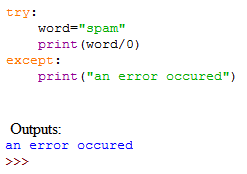


Untuk memahami cara kerja **try..except**, perhatikan contoh dibawah:



Block **try** mengandung code yang akan dites keberadaan exceptionnya. Jika ditemukan exception, maka block **try** akan dihentikan dan block **except** akan dijalankan. Jika tidak ditemukan exception, maka block **except** dihiraukan.

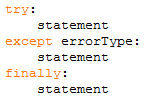
**except** dapat digunakan tanpa diberi error type untuk sekaligus menangkap semua jenis **exception**, seperti contoh berikut:



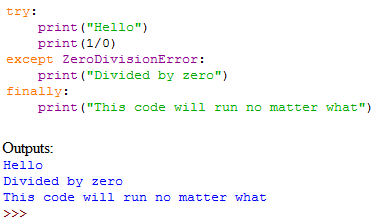
Namun, cara ini tidak disarankan karena dapat menyembunyikan kesalahan penulisan program. Contohnya, ketika menginspeksi code di atas, mungkin sekilas terlihat seperti **ZeroDivisionError**, namun sebenarnya terjadi **TypeError** yang disebabkan pembagian string **word** dengan integer **0**.

**finally:**

Untuk memastikan program berjalan tanpa mempedulikan error apapun yang ditemukan, dapat digunakan **finally**. **finally** dapat digunakan dengan menempatkannya di bagian paling bawah statement **try..except**. Berikut adalah format penggunaan **finally**:



Berikut adalah contoh dari penggunaan **finally**:



Seperti yang dapat dilihat di atas, **finally** terbukti pasti akan dijalankan diakhir. **finally** juga akan tetap berjalan meskipun ada **exception** yang tidak ditangkap oleh **except**.

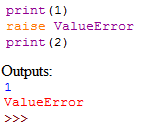
**raise:**

Ada pula **raise** yang dapat digunakan ketika programmer ingin menampilkan error type secara eksplisit.

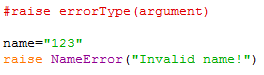
Berikut format **raise**:



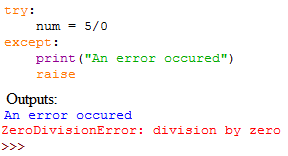
Contoh cara kerja **raise**:



**raise** dapat juga diberi argumen untuk memberi detil tambahan:



Di dalam block **except**, **raise** dapat ditambahkan (hanya **raise,** tanpa errorType) untuk me-**raise** kembali error yang bersangkutan. Contohnya seperti berikut:



1. Event Handling

*Event* adalah sebuah pemberitahuan terjadinya sesuatu yang spesifik, seperti mouse click pada sebuah gambar tombol.

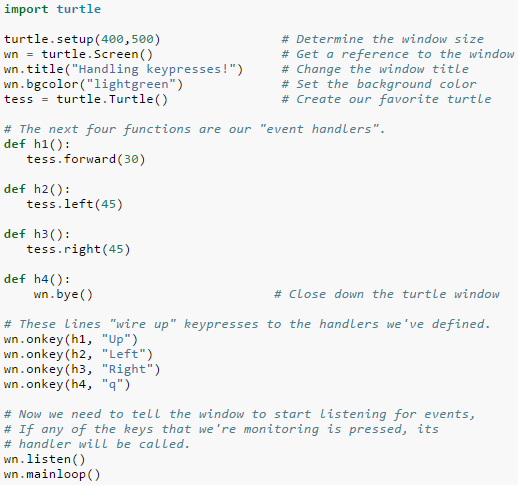
*Event Handler* adalah bagian code yang dijalankan ketika terdeteksi adanya sebuah *Event*.

Python mensupport pembuatan program berorientasi **Event** dengan GUI.

Dalam praktik, terdapat 2 komponen untuk **Event Handling**:

* Listener (Method yang akan mendeteksi **Event**)
* Event Handler (Menentukan function yang dijalankan berdasarkan **Event** yang terjadi)

Berikut adalah contoh kutipan code untuk **Event Handling**, memanfaatkan GUI, yang bereaksi terhadap tekanan tombol pada keyboard:



(Source: http://openbookproject.net/thinkcs/python/english3e/events.html)

Perhatikan bagian-bagian dari contoh code di atas. Code tersebut mengandung 2 komponen penting untuk membuat **Event Handler**.

1. Listener:

wn.listen()

1. Event Handler:

wn.onkey(h1, "Up")

Dimana h1 adalah nama function yang dipanggil, dan "Up" adalah tombol keyboard yang bersangkutan.

# BAB 12

**FUNCTIONAL PROGRAMMING LANGUAGES**

Jenis bahasa pemrograman yang sejauh ini digunakan adalah bahasa *imperative* yang desainnya berdasarkan arsitektur *von Neumann*.

Ada pula jenis bahasa pemrograman lain, yaitu: *functional programming language*. Aplikasi umumnya dapat dilihat pada function-function matematika.

Functional programming memiliki sejarah yang panjang, diawali dengan Lisp pada tahun 1958. Bentuk pemrograman ini biasanya digunakan dalam perdagangan, perusahaan algoritmik, serta untuk telekomunikasi.

1. Ciri-ciri bahasa pemrograman fungsional:

* Semuanya merupakan function
* Function *murni* tanpa "side effect"/merubah data aslinya
* Struktur data *immutable* (tidak dapat diubah setelah dibentuk)
* Menyimpan *state* dalam function
* Menggunakan rekursi, bukan iterasi/loop

1. Keuntungan Functional Programming:

* Tidak adanya *side effect* dapat memperkokoh program
* Program cenderung menjadi lebih modular dan terdapat dalam block-block penyusun yang kecil
* Sangat menekankan algoritma
* Cocok untuk parallel/concurrent programming dari segi konsep

1. Kekurangan Functional Programming:

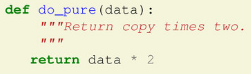
* Pemecahan permasalahan yang sama akan tampak sangat berbeda dibanding prosedural atau object-oriented
* Menemukan developer-developer yang mahir kadang sulit
* Tidak selalu berguna untuk berbagai macam masalah
* I/O termasuk *side effect* dan memerlukan penanganan khusus
* Rekursi jauh lebih rumit dibanding loop/iterasi
* Struktur data yang *immutable* dapat memperlambat run time

1. Pure Function

Pure Function adalah function yang tidak memiliki *side effect*. Function dalam hal ini tidak memodifikasi data, hanya men-*return* duplikat dari data asli yang telah dimanipulasi.

Pure Function dalam hal ini bukan benar-benar *pure* melainkan hanya kesepakatan bersama.

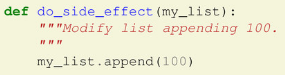
Perhatikan contoh *pure function* berikut, dapat dilihat bahwa **data** tidak dimanipulasi secara langsung sehingga tidak menghasilkan *side effect*:



1. Side Effect

Side Effect adalah manipulasi atau perubahan yang terjadi pada data di dalam function.

Berikut adalah contoh function yang menghasilkan *side effect*, dapat dilihat bahwa **my\_list** akan termodifikasi dengan function tersebut karena di-append:

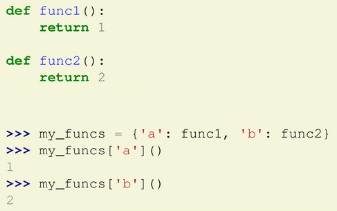


1. Using Functions

* Function sebagai Object:

Function dapat dijadikan object untuk meningkatkan dinamika dan keindahan program.

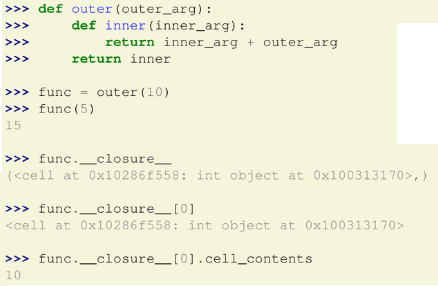
Berikut adalah contoh penggunaan function sebagai object:



* Closure:

Closure menyimpan *state* dari function.

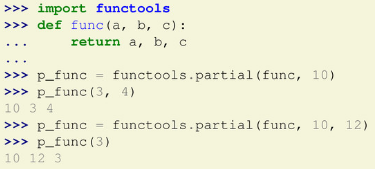
Berikut contoh penggunaan **\_closure\_**:



* functools & partial:

**partial** mengambil sebuah function sebagai parameter pertamanya. Lalu parameter selanjutnya merupakan bagian dari parameter function itu sendiri yang akan diberi nilai absolut/fixed value.

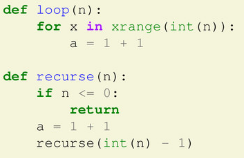
Perhatikan contoh penggunaan **functools** & **partial** berikut:



* Recursion

Seperti yang telah dibahas sebelumnya, rekursi adalah ketika function memanggil dirinya sendiri. Recursion dapat digunakan untuk iterasi function dalam Functional Programming.

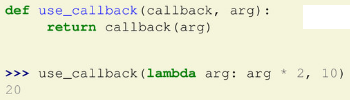
Berikut contoh Recursion:



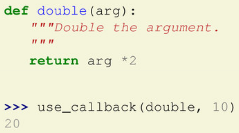
1. Lambda

Lambda dapat membantu untuk membuat function-function anonim (function yang tidak dibind dengan identifier). Dimana untuk menggunakan **lambda** hanya diperlukan expression. Sempat ada perundingan untuk menghapusnya dari Python 3, namun akhirnya **lambda** dipertahankan.

**lambda** sangat beguna untuk callback, perhatikan contoh berikut, dapat dilihat **arg\*2** adalah return value akhirnya:



Namun, **lambda** tidak wajib digunakan, berikut adalah alternatif jika tidak menggunakan **lambda**:



Dan masih ada banyak lagi cara membuat program berorientasi *Functional Programming*. Jadi pada dasarnya, programming berorientasi *Functional Programming* adalah metode programming yang menghindari *side effect*.

Berikut adalah link sumber untuk materi selengkapnya yang dibuat oleh Mike Müller mengenai *Functional Programming* untuk Python:

https://speakerdeck.com/pyconslides/functional-programming-with-python-by-dr-dot-ing-mike-muller

**BAB 13**

**LOGIC PROGRAMMING LANGUAGES**

Logic Programming Language

* Programming dalam bahasa logika atau *Programming in Logic* dapat di ekspresikan dengan bentuk simbol – simbol logika. Penarikan kesimpulan menggunakan logika merupakan proses untuk menghasilkan sebuah produk yang bersifat deklaratif.
* Berikut beberapa fungsi simbol – simbol logika atau yang dapat disebut dengan *predicate calculus* :
  1. Mengekspresikan preposisi dan hubungan antara 1 preposisi dengan yang lain
  2. Menjelaskan bagaimana kemunculan 1 preposisi baru dapat mempengaruhi preposisi yang lain
  3. Memberi tambahan kemampuan untuk merepresentasikan pengetahuan dengan lebih sebuah cermat dan detil
* Compund Terms
  1. Compound term adalah penulisan fungsi matematika yang berfungsi untuk menunjukan hubungan antara elemen dengan elemen lainnya.
  2. Contoh compound terms :
     + print (a)
     + input (b)
     + delete (c)

Dari contoh – contoh diatas dapat diidentifikasi bahwa compound terms terbagi menjadi 2 bagian penting, yaitu :

* + - * Functor  
        Simbol atau penanda dari function yang hanya menjadi nama dari hubungan antar elemen tersebut. Pada contoh (print, input, delete)
      * Parameter  
        Isi dari hubungan yang dibentuk. Dengan kata lain parameter adalah elemen – elemen yang dihubungkan oleh fungsi. Pada contoh ditulis (a, b, c)
* Programming in Logic
  1. *Programming in Logic* atau yang biasa disingkat *ProLog* adalah pemrograman logika. Dimana yang digunakan dalam pemrograman ini adalah bahasa yang deklaratif yang akan memberikan hasil akhir berupa kesimpulan. Selain itu *ProLog* lebih menggunakan relasi. Sehingga berbeda dengan bahasa – bahasa lain atau pemrograman – pemrograman yang lainnya.
* Simbol – simbol penting di *ProLog*

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Simbol** | **Contoh Penggunaan** | **Arti** |
| Negasi | ¬ | ¬ a | Bukan a |
| Konjungsi | ∩ | a ∩ b | a dan b |
| Disjungsi | ∪ | a ∪ b | a atau b |
| Equivalen | ≡ | a ≡ b | a senilai atau sama dengan b |
| Implikasi | ⊃  ⊂ | a ⊃ b  a ⊂ b | a bagian dari b  b bagian dari a |
| Universal | ∀ | ∀A . P | P benar untuk semua A |
| Sebagian | ∃ | ∃B . Q | Q benar untuk sebagia B |

* Dasar pemrograman *ProLog*
  1. Fakta  
     Kenyataan yang ingin disampaikan melalui *ProLog* yang menyatakan relasi antara 2 elemen

Contoh : Herman adalah ayah Fikri. Ditulis kedalam *ProLog* menjadi  
Ayah(Herman, Fikri)

* 1. Aturan  
     Relasi yang terbentuk melalui implikasi atau sebab – akibat

Contoh : Gerald adalah ayah Herman, oleh karena itu Gerald adalah kakek dari Fikri. Ditulis kedalam *ProLog* menjadi

Kakek(Gerald, Fikri) => Ayah(Gerald, Herman)

* 1. Klausa  
     Klausa berupa tubuh dari sebuah aturan yang dipisahkan dengan tanda “:-”. Klausa merupakan susunan kata dalam sebuah *ProLog* yang diakhiri dengan titik. Klausa dapat terdiri dari beberapa sub-klausa yang dihubungkan dengan tanda “,” untuk mewakili dan(konjungsi) sedangkan “;” untuk mewakili atau(disjungsi).  
     Contoh : Gerald ayah Herman. Herman ayah Fikri. Ditulis kedalam *ProLog* menjadi

Kakek(Gerald, Fikri) => Bapak(Gerald, Herman) , OrangTua(Herman, Fikri)

* 1. Relasi  
     Istilah – istilah yang kita gunakan selama membuat fakta maupun aturan seperti, *Bapak / Kakek / OrangTua* disebut sebagai relasi. Relasi berbentuk sebuah tabel yang memuat beberapa baris fakta atau aturan. Contoh :

|  |  |  |
| --- | --- | --- |
| [a][b][a,b] | [a,b][c,d][a,b,c,d] | [a,b][c][a,b,c] |

* 1. Variabel  
     Variabel dalam *ProLog* disebut juga saebagai term. Dimana variabel disini dibagi menjadi 2, yaitu :
     + Variabel bernama
     + Variabel tak bernama (Placeholder) biasa dilambangkan “\_”

Variabel harus diawali dengan huruf besar atau huruf kapital. Karena jika menggunakan huruf kecil dianggap adalah sebuah relasi. Selain itu Variabel tak bernama juga biasa digunakan untuk mengabaikan nilai dari variabel tersebut.

**DAFTAR PUSTAKA**

1. BAB 1 Introduction

* https://kbp13uksw.wordpress.com/2009/06/19/syntax-semantic-grammar/
* https://sultanluckq.wordpress.com/2013/11/09/syntax-semantic/
* http://nustaffsite.gunadarma.ac.id/blog/tanzir/2007/10/06/konsep-bahasa-pemrograman-sintaks-semantik-dan-pragmatik/

1. BAB 3 Naming, Bindings and Scopes:

* https://pythonconquerstheuniverse.wordpress.com/2009/10/03/static-vs-dynamic-typing-of-programming-languages/ (for Binding)
* http://python-textbok.readthedocs.io/en/1.0/Variables\_and\_Scope.html (for Scopes)

1. BAB 5 Expression and Assignment Statements

* http://chii90.wordpress.com/2013/04/06/chapter-7-expressions-and-assignment-statements-review-questions/
* http://arcana70.blogspot.co.id/2014/03/kbp-pert-5\_23.html
* https://docs.python.org/3/reference/expressions.html
* https://docs.python.org/3/reference/simple\_stmts.html

1. BAB 7 Subprograms

* https://docs.python.org/3/faq/programming.html#how-do-i-write-a-function-with-output-parameters-call-by-reference
* https://www.quora.com/What-are-the-differences-between-shallow-binding-deep-binding-and-ad-hoc-binding
* http://arcana70.blogspot.co.id/2014/04/kbp-pert-7.html
* https://docs.python.org/3/faq/programming.html
* https://docs.python.org/3/library/asyncio-task.html
* http://sakti.github.io/python101/fungsi.html

1. BAB 9 Object-Oriented Programming

* http://thepythonguru.com/python-inheritance-and-polymorphism

1. BAB 11 Exception and Event Handling

* http://openbookproject.net/thinkcs/python/english3e/events.html

1. BAB 12 Functional Programming Language

* https://speakerdeck.com/pyconslides/functional-programming-with-python-by-dr-dot-ing-mike-muller

1. BAB 13 Logic Programming Language

* http://arcana70.blogspot.co.id/2014/06/kbp-pert-13.html
* http://www.slideshare.net/OtbiePutra/if412-07
* https://setiawanandy66.wordpress.com/2013/06/28/kbp-chapter-16/
* https://kmahfud.wordpress.com/2016/01/31/propositional-logic-dan-predicate-calculus/
* http://heriyantolie.blogspot.co.id/2011/07/rangkuman-kbp.html

# LINK BLOG ANGGOTA KELOMPOK 1 (PYTHON)

* GERALD : http://gerosama.blog.binusian.org/
* HANSEN : http://coposaja.blog.binusian.org/
* FELIX : http://felixwenalvin.blog.binusian.org/
* ALVIN : http://springbreezever.blog.binusian.org/
* FIKRUL : http://frixs.blog.binusian.org