**ALGORITMA DAN PEMROGRAMAN**

**JOBSHEET PROGRAM PYTHON**



Mata Kuliah : Algoritma dan pemrograman  
 Dosen : Prayitno, S.ST.,M.T,Ph.D

Disusun oleh

Nama : Maulana Azka Rifki Saputra

NIM : 3.34.24.1.13

**PROGRAM STUDI TEKNIK INFORMATIKA**

**JURUSAN TEKNIK ELEKTRO**

**POLITEKNIK NEGERI SEMARANG**

**2024**

**PROGRAM**

Sebuah program adalah lebih dari sekadar daftar pernyataan aritmatika atau yang serupa yang dieksekusi secara berurutan. Apa yang membuat komputer begitu kuat adalah kemampuan mereka untuk mengontrol urutan operasi. "Kuat" harus dipahami dalam arti kekuatan komputasi, yaitu kemampuan untuk melakukan perhitungan kompleks, bukan dalam arti kecepatan komputasi. Seseorang mungkin ingin mengeksekusi beberapa bagian dari program tergantung pada suatu kondisi, atau mengulangi suatu bagian beberapa kali, dll. Bahasa tingkat tinggi seperti PYTHON menyediakan beberapa instruksi untuk memenuhi kebutuhan ini. Instruksi-instruksi ini termasuk dalam kumpulan pernyataan aliran kontrol.

Jenis fungsionalitas penting lainnya adalah instruksi input/output. Dengan sendirinya, sebuah program yang tidak berinteraksi dengan lingkungannya tidak memiliki minat (itu akan menjadi sistem tertutup yang hanya mengonsumsi daya prosesor dan menghasilkan panas tanpa manfaat). Tidak ada program yang tidak berinteraksi dengan lingkungannya dalam beberapa cara selama eksekusinya, seperti menghasilkan data dalam file, mencetak nilai, meminta input, mengirim nilai di jaringan, dll.

Dalam bab ini, kita akan membahas proses menulis program dengan fokus utama pada aliran kontrol.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**3.1 PROGRAM**

Dalam PYTHON, sebuah program (atau skrip) disimpan dalam setidaknya satu file sumber, yaitu file teks sederhana yang isinya sesuai dengan sintaks PYTHON. Meskipun tidak wajib, sangat disarankan agar nama file sumber PYTHON diakhiri dengan .py (misalnya myprogram.py atau myapp.py).

Menulis file sumber tidak berkaitan langsung dengan eksekusinya. Kita menulis program tersebut dan kemudian mengeksekusinya saat diperlukan.

Untuk menulis program, kita dapat menggunakan editor teks apa pun yang kita suka untuk menghasilkan file sumber. Ada banyak IDE yang akan sangat membantu Anda (misalnya PYCHARM, atau VISUAL STUDIO CODE, lihat Bab 2).

► Jangan bingung antara editor teks dengan pengolah kata. Editor teks adalah alat untuk mengedit file teks murni, file yang hanya berisi urutan karakter. Pengolah kata adalah alat untuk mengedit dan menata teks agar dapat ditampilkan dengan gaya. Sebagai contoh:

* VIM, EMACS adalah editor dasar, tampilannya sangat sederhana karena antarmukanya berupa teks murni.
* PYCHARM atau VISUAL STUDIO CODE adalah editor teks dengan antarmuka grafis yang memiliki banyak fitur untuk membantu menulis kode PYTHON.
* MS-WORD, LIBREOFFICE, GOOGLE DOCS adalah pengolah kata, jangan gunakan mereka untuk menulis program, Anda tidak akan berhasil.

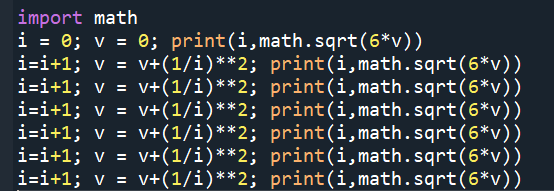
Setelah file sumber dibuat, kita bisa menjalankan program yang ada di dalamnya dengan memberikannya kepada penerjemah PYTHON. Penerjemah PYTHON berperan dalam mode non-interaktif atau mode skrip (berbeda dengan mode REPL) dari PYTHON yang menjalankan instruksi yang disediakan dalam kode sumber. Tentu saja, file sumber dapat dieksekusi kapan pun kita mau, menjadikannya sebuah aplikasi (istilah tidak formal yang digunakan untuk menyebut sesuatu yang dapat Anda jalankan).

**3.2 Sequence**

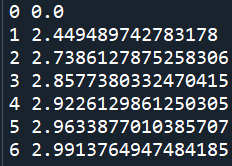
Dari sekarang, kita bisa menulis pernyataan apa pun yang sudah kita ketahui dalam file sumber. Ingat program pertama yang ditampilkan di pengantar bagian ini (lihat Program 1.1).

Sebagai contoh lain, kita bisa mencoba menulis program yang akan menghitung perkiraan π seperti yang disarankan dalam Latihan 2.12.

Program 3.2 : Approximation of π



Outputnya menjadi seperti ini :

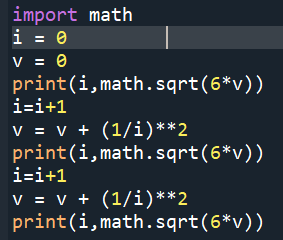


Flowchart :

<https://drive.google.com/file/d/1ci37nXxrrVeCDRa5Hz0FejUOwcTuQ2K3/view?usp=drive_link>

Meskipun valid, tidak disarankan untuk menggunakan operator sequencing ; (titik koma) terlalu  banyak dalam program. Satu pernyataan dalam satu baris jauh lebih mudah dibaca dan dikelola. Kita dapat lebih mudah menyisipkan pernyataan di antara dua pernyataan, menghapus pernyataan,  menambahkan komentar, dll. Jadi penulisan program yang lebih baik bisa seperti ini:

Program 3.3 Kode yang telah diformat



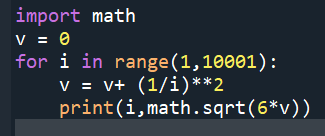
**3.3 Iteration, the for**

Mengamati Program 3.3 dan Program 3.4 mudah untuk melihat bahwa urutan tersebut mencakup kelompok baris yang semuanya persis sama: Dalam Program 3.3 baris 5 hingga 9 adalah replika dari baris 4; Dalam Program 3.4 kelompok baris 9-11, dan 12-14, dll adalah replika dari kelompok 5-7.

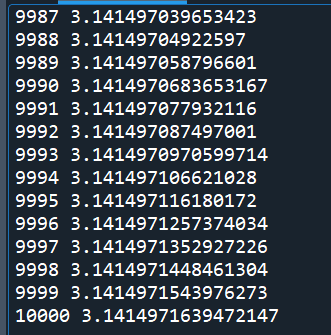
Gagasan utamanya adalah memiliki variabel (di sini i) yang akan mengambil waktu setelah waktu semua nilai dari 1 hingga 10000 (variabel ini mengontrol loop), dalam urutan tersebut; dan kemudian mengeksekusi untuk setiap nilai bagian kode yang sesuai. Seluruh iterasi terjadi di kotak abu-abu terang dan bagian kode yang ingin kita ulangi ada di kotak yang lebih gelap. Apa yang ada di dalam kotak terang dan tidak ada di kotak gelap adalah struktur kontrol dari loop, yang "diotomatisasi" oleh python ketika kita menggunakan for-loop seperti itu; ini adalah jenis hal yang ditawarkan oleh bahasa tingkat tinggi.

Ada banyak variasi berbeda dari iterasi for di python, tetapi kami hanya akan memperkenalkan beberapa lagi dalam alur buku. Sebuah for loop bisa (kami secara eksplisit menggambar karakter spasi untuk menunjukkan strukturasi kode):

Program 3.4 : Iterations with a range



Dan menghasilkan output seperti ini :



Flowchart :

<https://drive.google.com/file/d/1fILzOKD2EfrECSs2K31LJ620PHtdjh3F/view?usp=drive_link>

Catatan bahwa wajib memiliki tanda titik dua (:) di akhir baris dengan for untuk memperkenalkan blok bersarang. Penyusunan ditunjukkan dengan penggunaan indentasi kode. Karakter spasi menunjukkan indentasi. Indentasi tersebut wajib, itu adalah bagian dari sintaks python. Ini penting, kita perlu menunjukkan apa yang ingin kita ulangi, seperti yang diilustrasikan oleh diagram alir. Indentasi mencerminkan apa yang digambar dalam diagram alir, blok di dalam iterasi berada di sebelah kanan, sehingga diindentasikan ke kanan dalam kode. Hati-hati, sementara dalam diagram alir, posisi node tidak signifikan, sintaks python mengharuskan blok yang akan diulang harus diindentasikan ke kanan. Dalam diagram alir, seluruh for loop berada di kotak abu-abu terang; dan blok yang diiterasi berada di kotak abu-abu gelap. Seseorang juga dapat mencatat bahwa ada panah mundur dalam diagram alir, inilah yang membuat loop :

* dimulai dengan baris for i in range(1,10001): yang membentuk mekanisme kotak abu-abu terang;
* dan dua baris terindentasi berikutnya yang membentuk blok kode yang akan diulang (kotak abu-abu gelap).

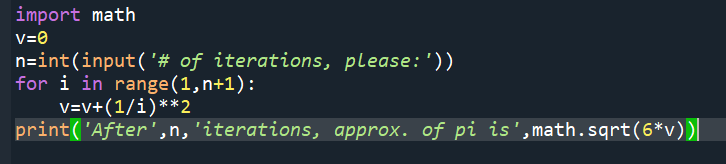
Perlu diketahui juga bahwa apa pun yang kita gunakan untuk indentasi harus seragam di sepanjang blok yang diindentasi. Di sini, kita menggunakan empat spasi pada baris 5 dan 6. Kita dapat menggunakan 1 spasi, 23 spasi, satu tab, atau 3 spasi dan satu tab, dst., asalkan kita konsisten di seluruh blok. Sekali lagi jangan tertipu, selalu gunakan jenis indentasi yang sama dalam program tertentu, yang umum adalah 4 spasi, atau 1 tab. Setiap indentasi yang salah atau tidak konsisten akan dilaporkan sebagai kesalahan, PYTHON memaksakan struktur alur agar tercermin secara tepat ke dalam kode; di hampir semua bahasa lain, struktur diekspresikan dengan jauh lebih bebas.

Kami menggunakan kata kunci for untuk memperkenalkan loop, diikuti oleh variabel (variabel kontrol yang akan memilih nilainya dalam suatu rentang dengan menggunakan fungsi range), dan mengeksekusi blok kode yang menjorok yang mengikutinya. Rentang PYTHON seperti rentang (a, b) adalah rentang setengah terbuka, yaitu [a, b), dan mewakili dalam urutan menaik semua nilai integer dari a hingga b (dikecualikan) satu per satu.

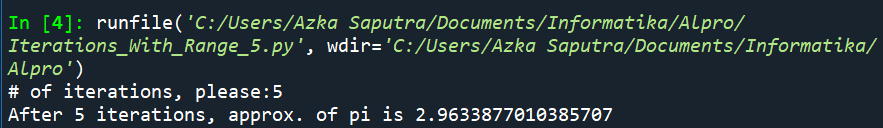
**3.4 Input**

Bagaimana jika kita ingin menghitung pendekatan ke-10? Ke-1000? Tentu kita bisa mengedit kode sumber untuk setiap kebutuhan, tetapi salah satu solusi adalah memikirkan ulang masalahnya. Mengapa kita tidak meminta input dari pengguna? Itulah tujuan dari fungsi input.

Program 3.5 : Interactive approximation of π



Lalu outputnya seperti ini :



Flowchart :

[**https://drive.google.com/file/d/1KArbVkQ9ALiYW1VmwEyoYQuf4oHoRvzT/view?usp=drive\_link**](https://drive.google.com/file/d/1KArbVkQ9ALiYW1VmwEyoYQuf4oHoRvzT/view?usp=drive_link)

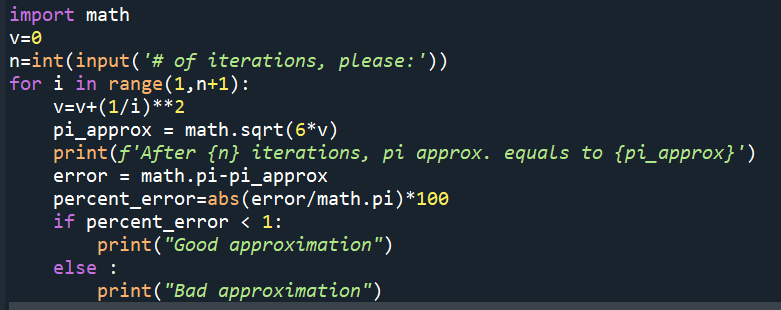
Fungsi input mengambil pesan (sebuah string) sebagai parameter dan ketika dieksekusi mencetak pesan tersebut di konsol, kemudian menunggu pengguna untuk mengetik sesuatu dengan keyboard dan mengembalikan string yang dibaca. Karena kita menginginkan nilai sebagai int, kita kemudian mengonversi string tersebut.

**3.5 Strings**

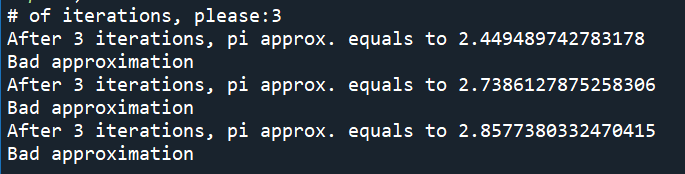
**3.6 Alternate, the if**

Bentuk kedua ini digunakan jika kita tidak memiliki hal khusus yang perlu dilakukan dalam bagian "false". Catat bahwa, seperti pada loop for, baris if dan else diakhiri dengan : (titik dua) untuk memperkenalkan blok instruksi masing-masing. Sama seperti pada loop for, blok bagian dalam ("true"/"false") harus diindentasi ke kanan. Yang penting untuk dicatat adalah bahwa blok harus diindentasi ke kanan dari if dan else (jika ada).

Jadi kita bisa menulis program aproksimasi π seperti ini:



Dan mengeluarkan output seperti ini :



Flowchart :

<https://drive.google.com/file/d/18nz8yYeR4AYwbUAXObXRHlGe_SE_bocB/view?usp=drive_link>

**3.7 Conditional Loop, the while**

Dalam beberapa program sebelumnya, kami meminta pengguna untuk memasukkan sebuah nilai, tetapi bagaimana jika nilai yang diberikan salah? Pada titik ini, kami harus mempertimbangkan dua jenis kesalahan:

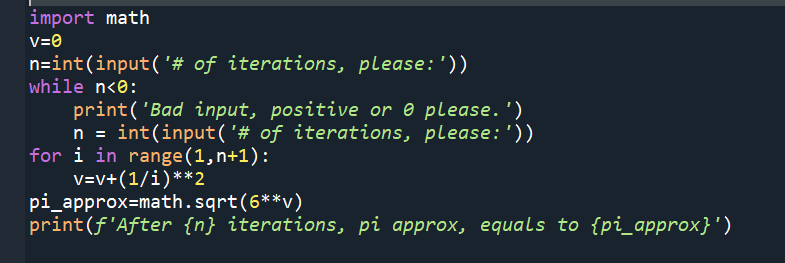
* Nilai yang salah dari sudut pandang tipe (pengguna memasukkan string di mana yang diharapkan adalah integer)
* Nilai yang salah dari logika perhitungan (pengguna memasukkan nilai negatif di mana yang diharapkan adalah nilai positif).

Secara umum, kita ingin terus meminta pengguna untuk memasukkan nilai sampai dia memberikan yang benar (memiliki input yang benar adalah kondisi yang diperlukan untuk melanjutkan perhitungan π, bukan?).

Untuk jenis kesalahan pertama, nanti kita akan melihat bagaimana menangkapnya (lihat Bab 4.1.5) karena terkait dengan struktur aliran kontrol.

Untuk kasus kedua, kita hanya perlu mengulangi pertanyaan sampai input benar atau, secara simetris, selama input salah. Banyak bahasa pemrograman menawarkan loop while dan until. PYTHON hanya memiliki satu (karena mereka sangat simetris, hanya satu yang cukup), loop while menggunakan kata kunci while:

Program 3.14 Input validation



Lalu menghasilkan output seperti :



Flowchart :

<https://drive.google.com/file/d/14S-k1G9c6o0MAC-OUBVoxr_oBYOQ8ntp/view?usp=drive_link>

**3.8 Early Exits and Continuations in Loops**

Kami terkadang perlu keluar dari loop lebih awal atau untuk memotong aliran di dalam blok bagian dalam. Meskipun konstruksi semacam itu tidak wajib, banyak bahasa pemrograman menawarkan instruksi semacam itu karena akan sangat memudahkan penulisan/pembacaan program. Tanpa itu, kondisi dan blok akan jauh lebih kompleks untuk diekspresikan.

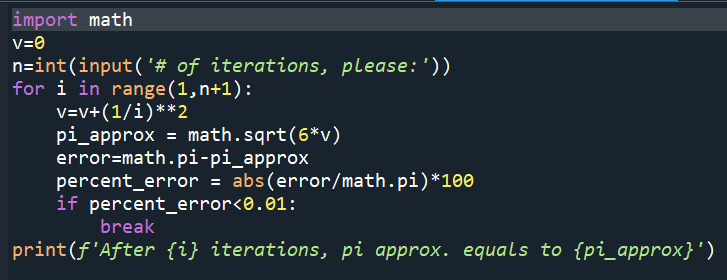
Instruksi-instruksi ini terkait dengan goto yang (tidak) terkenal. Ada makalah terkenal dari ilmuwan komputer Edgar Dijkstra, berjudul "Goto considered Harmful" yang menjelaskan mengapa melompati kode secara tidak terkendali adalah praktik yang sangat buruk yang mengarah pada "kode Spaghetti". Jenis kode yang aliran eksekusinya terlalu kompleks untuk dipahami dan hampir tidak dapat dibaca atau sulit dipelihara. Setidaknya bagus untuk mengetahui bahwa ada lompatan non-kondisional yang tidak berbahaya dalam bentuk instruksi break dan continue.

**3.8.1 Early Exit, the break**

Bagaimana jika kita ingin mendekati nilai π sedemikian rupa sehingga kita mencapai jumlah total iterasi yang diharapkan atau ketika perkiraan sementara sudah cukup baik? Dalam loop mana pun kita dapat menggunakan perintah break untuk keluar dari loop segera dan memberikan kontrol kepada instruksi berikutnya setelah loop (instruksi berikutnya pada tingkat yang sama dengan loop). Diagram alir untuk perintah break bisa seperti yang ada di Gambar 3.7.

Dengan demikian, kita dapat menulis program yang berhenti setelah sejumlah iterasi tertentu atau jika perkiraannya sudah cukup baik :

Program 3.15 : break



Lalu menghasilkan output seperti ini :

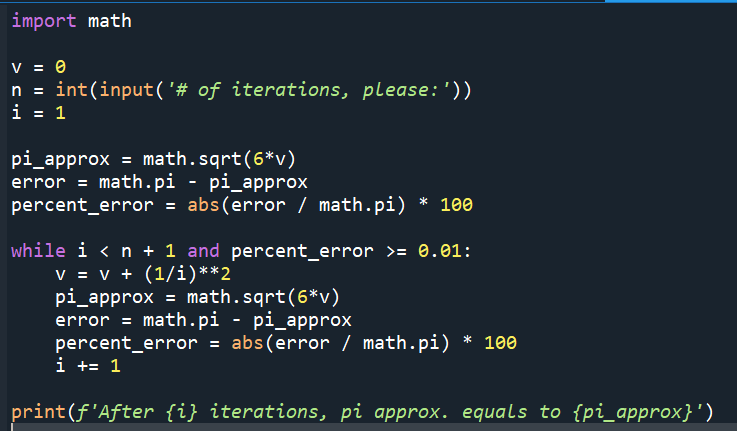


Flowchart :

<https://drive.google.com/file/d/1Kj3xCK02fJZc_FlQOwJ6dJpQmbDrmDWu/view?usp=drive_link>

Di dalam for loop, kita menghitung kesalahan relatif, dan ketika kesalahan ini kurang dari kriteria kita, kita "menghentikan" loop dengan mengeksekusi perintah break. Ini mudah untuk ditulis dan dipahami. Tanpa instruksi seperti itu, akan jauh lebih sulit untuk mendapatkan perhitungan yang sama. Kita harus menentukan nilai terkecil dalam rentang yang memenuhi kondisi atau mencampur kedua kondisi dalam loop while. Secara umum dan dalam kasus pertama, mungkin tidak mungkin atau setidaknya tidak mudah untuk menentukan nilai tersebut! Kasus kedua dapat ditulis seperti ini:

Program 3.16: While untuk mengganti break



Lalu menghasilkan output seperti :

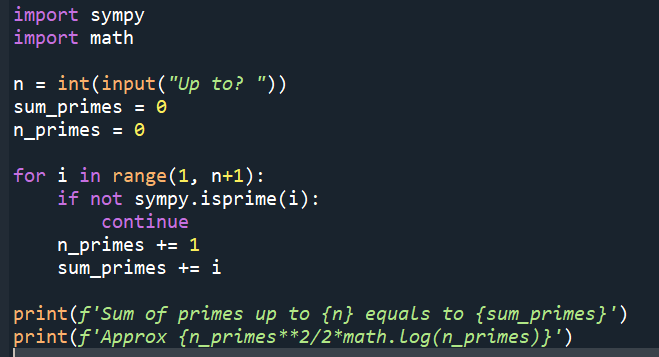


Flowchart : <https://drive.google.com/file/d/1AQX8FDlZxPCONdQSif51sgwhZD1G7jPe/view?usp=drive_link>

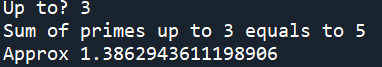
**3.8.2 Early Continuation, the continue**

Dalam beberapa kasus lain, kita ingin melewati beberapa bagian dari blok iterasi yang lebih dalam atau mengambil jalur pintas untuk beberapa iterasi. Misalnya, kita ingin menjumlahkan semua bilangan prima hingga angka tertentu *n*. Solusi sederhana adalah dengan mengiterasi semua angka dari 2 hingga *n* dan melewati angka yang bukan bilangan prima. Proses melewati ini diekspresikan dengan perintah continue yang mengabaikan sisa blok dan langsung melompat ke iterasi berikutnya dari loop. Sebuah diagram alir diberikan pada Gambar 3.8.

Program 3.17: Sum of prime numbers



Lalu menghasilkan output seperti :



Flowchart :

<https://drive.google.com/file/d/196G_3R065sMmA2mMLaeM8DHWj2WhvPiw/view?usp=drive_link>