## Pengantar Algoritma Rekursif

[](https://www.dicoding.com/academies/620/tutorials/35118?from=35113)Yuhu! Selamat datang pada materi terakhir dalam kelas ini. Tak terasa, kita telah mengarungi berbagai materi tentang pemrograman prosedural, mulai dari mengenal program dan bahasa pemrograman hingga berkutat dengan kumpulan data. Tentunya proses belajar tersebut tidaklah mudah. Oleh karena itu, Anda harus bangga dengan diri sendiri karena berhasil pada tahap ini.

Pada materi sebelumnya, Anda sudah mengulik lebih dalam terkait kumpulan data. Mulai dari konsep data yang variatif dan bersarang, metode pencarian data, serta algoritma pengurutan data telah Anda pelajari.

Pada materi kali ini, kita akan menelaah cara kerja algoritma rekursif secara komprehensif. Untuk itu, Anda perlu belajar beberapa materi mengenai hal berikut:

* pengertian algoritma rekursif dalam pemrograman;
* komponen penting dalam algoritma rekursif;
* jenis algoritma rekursif;
* kelebihan dan kekurangan algoritma rekursif; serta
* penerapan tingkat lanjut dalam menjalankan algoritma rekursif.

Sebagaimana materi sebelumnya, kita lakukan senam otak terlebih dahulu, yuk. Aktivitas ini membawa Anda lebih mudah menggambarkan materi yang akan dibahas. Dengan begitu, Anda dapat membayangkan dan memperkirakan cara kerja algoritma rekursif ini. Mari kita amati analogi di bawah ini, ya.

Ibu Fika ingin memasak capcai. Di dalamnya berisi beberapa sayuran, seperti wortel, buncis, brokoli, dan sawi putih. Ibu perlu menyiapkan beberapa sayuran tersebut untuk melengkapi isi dari masakan capcai.

Sebelum dimasak, Ibu butuh memotong sayuran supaya lebih mudah untuk mengunyah. Caranya, Ibu mengambil salah satu sayuran dan memotongnya menjadi dua bagian. Kemudian, setiap setengah potong sayuran tersebut dipotong menjadi dua bagian yang lebih kecil lagi. Ibu terus melakukan hal tersebut hingga potongan sayuran memiliki lebar yang cukup kecil.

[](https://www.dicoding.com/academies/620/tutorials/35118?from=35113)

Aksi memotong yang dilakukan Ibu Fika secara berulang tersebut adalah teknik rekursif. Setiap aksi tersebut diulang terus-menerus dengan masukan yang berasal dari output proses sebelumnya. Aksi memotong akan berhenti ketika mencapai kondisi tertentu, yakni potongan sulit dipegang oleh tangan.

Dalam konteks pemrograman, algoritma rekursif adalah suatu fungsi atau prosedur yang berisi aksi untuk memanggil fungsi itu sendiri. Hal tersebut akan berulang terus-menerus hingga kondisi tertentu terpenuhi. Layaknya perulangan, Anda perlu memastikan algoritma ini memiliki kondisi berhenti agar tidak terjadi perulangan tak terbatas atau infinite loop.

Dalam dunia nyata, Anda dapat merasakan aksi sebuah algoritma rekursif. Pernahkah Anda berdiri di suatu tempat yang setiap sisinya memiliki cermin? Ya, Anda dapat melihat bayangan diri sendiri berulang-ulang hingga mata sulit melihat ujungnya. Hal tersebut adalah salah satu konsep rekursif karena bayangan Anda akan memantul secara bolak-balik antara cermin satu dengan lainnya.

[](https://www.dicoding.com/academies/620/tutorials/35118?from=35113)

Sumber: [Flickr: Unggahan joshtaiger](https://flickr.com/photos/44124323209@N01/273593601)

Aksi Ibu Fika dalam memotong sayur pun dapat dikatakan sebagai aksi rekursif. Mengapa demikian? Ketika Ibu Fika memotong sayur yang utuh menjadi dua bagian, beliau menjalankan aksi “memotong sayur hingga terbelah menjadi dua”. Kemudian, aksi tersebut berulang dengan mengambil potongan tersebut sebagai argumen sebuah fungsi. Serangkaian aksi rekursif ini akan berulang hingga batas yang ditentukan, yaitu jumlah atau ukuran potongan telah terpenuhi.

[](https://www.dicoding.com/academies/620/tutorials/35118?from=35113)

Apakah Anda dapat membayangkan cara kerja algoritma rekursif? Kali ini, kita akan belajar algoritma rekursif lebih mendalam, mulai dari cara kerja, kondisi berhenti, hingga contoh penerapannya. Penasaran? Tunggu apa lagi, yuk, segera ke materi berikutnya!

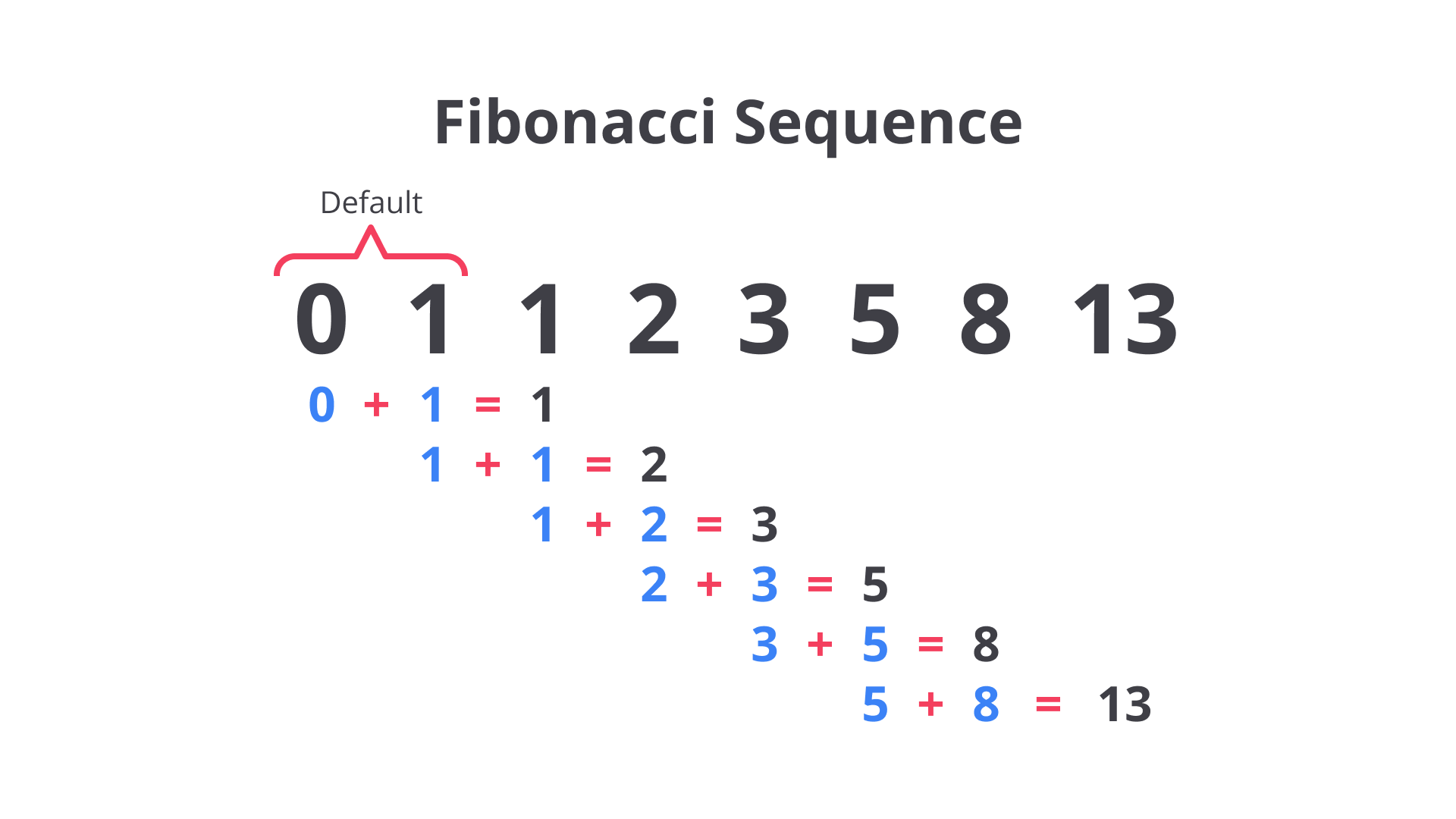
## Komponen Penyusun Algoritma Rekursif

|  |
| --- |
| 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, … |

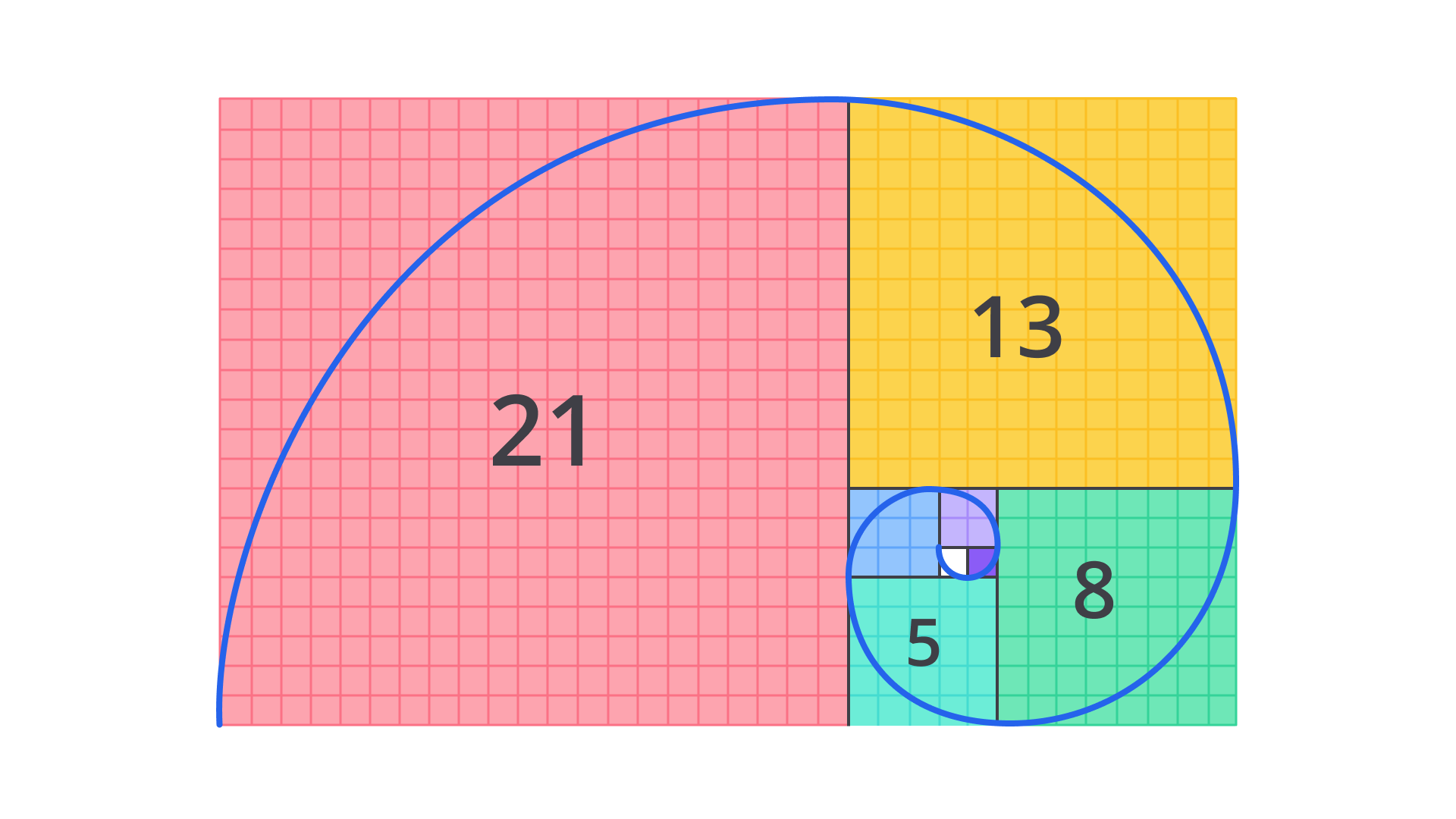
Tahukah Anda deret angka di atas? Ya, deret tersebut adalah deret fibonacci. Ia adalah sebuah deret bilangan yang sering kita temui dalam konteks keilmuan matematika.

Mengapa kita memulai materi dengan angka fibonacci? Deret fibonacci adalah salah satu penerapan algoritma rekursif. Ia dimulai dengan dua angka awal, yaitu **F(0) = 0**dan**F(1) = 1**, untuk melakukan proses rekursif. Kemudian, setiap bilangan berikutnya didapatkan dari hasil penjumlahan dari dua bilangan sebelumnya dalam urutan tersebut.

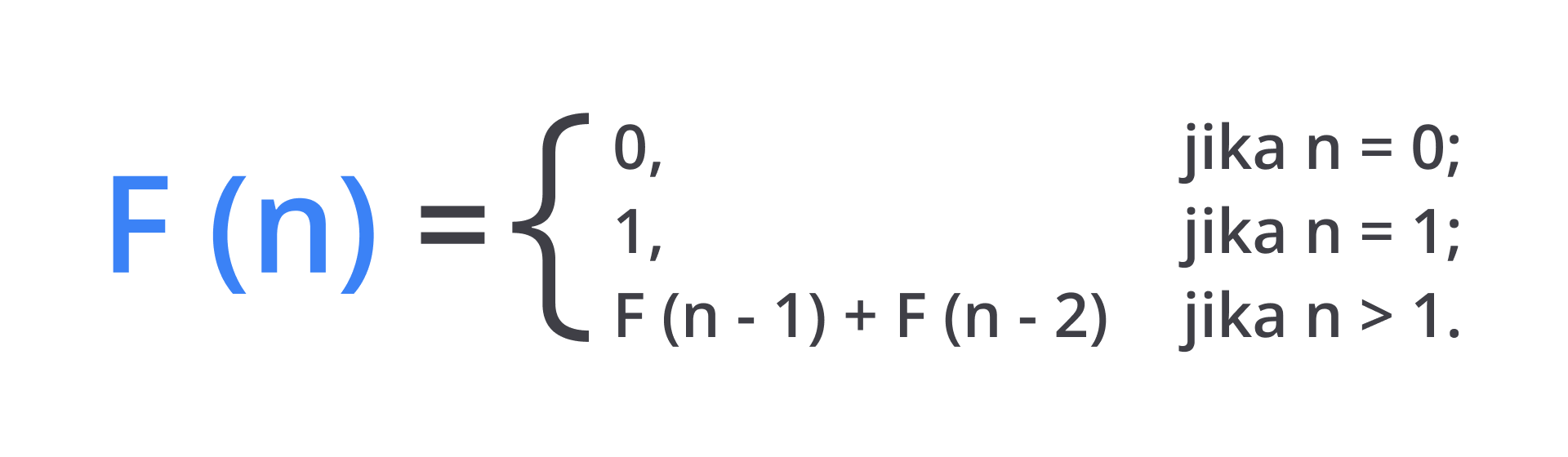
Misalnya, kita ingin mengetahui elemen indeks ke-2. Kita dapat menghitungnya dengan cara menjumlahkan angka pada indeks ke-0 dan ke-1 sehingga didapatkan perhitungan **F(2) = F(1) + F(0)**. Cara yang sama berlaku untuk elemen yang lain. Alhasil, kita mendapatkan pola perhitungan fibonacci untuk seluruh elemen fibonacci.

[](https://www.dicoding.com/academies/620/tutorials/35123?from=35118)

Biasanya, bilangan fibonacci juga digambarkan dalam bentuk spiral, yang dikenal juga sebagai golden spiral. Dengan perbandingan garis horizontal dan vertikal yang ukurannya sesuai dengan deret angka fibonacci, didapatkan sebuah pola. Pola spiral tersebut menciptakan angka-angka yang semakin besar saat bergerak keluar dari pusat spiral. Bentuk tersebut merepresentasikan hubungan visual antara angka dalam deret fibonacci dan bentuk geometris yang unik.

[](https://www.dicoding.com/academies/620/tutorials/35123?from=35118)

Berdasarkan pola tersebut, kita dapat menuliskan suatu persamaan fibonacci. Berikut formulanya.

[](https://www.dicoding.com/academies/620/tutorials/35123?from=35118)

### Proses Pemecahan Masalah

Mungkin, saat ini Anda memiliki pertanyaan, berapakah nilai elemen fibonacci ke-5? Berdasarkan formula di atas, kita tentu bisa menemukan elemen fibonacci dengan n lebih dari 1. Masalahnya, Anda memerlukan nilai dari elemen-elemen sebelumnya. Tidak serta merta elemen tersebut dapat ditentukan. Ada proses yang perlu dihitung untuk mencari nilai elemen yang diinginkan.

Untuk mencari elemen fibonacci indeks ke-5, Anda harus mengetahui elemen fibonacci pada indeks ke-4 dan ke-3. Kemudian, untuk mengetahui elemen indeks ke-4, Anda dapat menghitungnya dari elemen indeks ke-3 dan ke-2. Proses tersebut berlanjut hingga mengakses elemen indeks ke-1 dan ke-0.

* fibonacci(5) = fibonacci(4) + fibonacci(3)
* fibonacci(4) = fibonacci(3) + fibonacci(2)
* fibonacci(3) = fibonacci(2) + fibonacci(1)
* fibonacci(2) = fibonacci(1) + fibonacci(0)

Jadi, apabila ingin mencari nilai dari elemen fibonacci pada indeks selain 0 atau 1, Anda perlu mengurutkannya dari indeks ke-n-1 hingga indeks ke-0. Setelah indeks ke-2, yaitu 0 atau 1, pola perhitungan berubah tanpa menjumlahkan kedua elemen deret. Proses penjumlahan hanya berlaku untuk nilai n lebih dari 1.

Proses inilah yang perlu diperhatikan sebelum melakukan rekursif. Kita perlu memecah masalah yang besar menjadi lebih kecil dan serupa. Setiap masalah tersebut dapat diselesaikan dengan solusi yang identik atau sama.

Berdasarkan pola perhitungan di atas, kita dapat menuliskan sebuah program untuk mencari nilai dari elemen fibonacci. Ingat, program tidak perlu terikat dengan bahasa pemrograman. Anda hanya perlu menuliskan alur penyelesaian masalah supaya bisa dipahami oleh orang lain. Berikut adalah program untuk mencari elemen fibonacci.

|  |
| --- |
| 1. Tentukan elemen indeks **n** yang ingin dicari. 2. Apabila nilai **n** sama dengan 0, silakan kembalikan nilai 0. 3. Apabila nilai **n** sama dengan 1, silakan kembalikan nilai 1. 4. Apabila nilai **n** lebih dari 1, hitung dan kembalikan nilai fibonacci dengan formula elemen indeks ke-n-1 + elemen indeks ke-n-2. |

### Pembuatan Fungsi

“Recursion occurs when a function calls itself directly or indirectly.”

– [Mastering recursive programming - IBM Developer](https://developer.ibm.com/articles/l-recurs/)

Berdasarkan potongan kalimat di atas, rekursi dapat terjadi ketika suatu fungsi memanggil dirinya sendiri secara langsung atau tidak. Untuk itu, bungkuslah serangkaian aksi dengan fungsi supaya memudahkan kita dalam memanggil aksi yang berulang. Selain itu, berikan sebuah parameter untuk memudahkan kita mengubah nilai saat fungsi itu dipanggil. Jadi, program akan berubah menjadi seperti berikut.

|  |
| --- |
| 1. Tentukan elemen indeks **n** yang ingin dicari. 2. Buatlah fungsi bernama “fibonacci” dengan parameter **n** bertipe data integer positif.    1. Apabila nilai **n** sama dengan 0, silakan kembalikan nilai 0.    2. Apabila nilai **n** sama dengan 1, silakan kembalikan nilai 1.    3. Apabila nilai **n** lebih dari 1, hitung dan kembalikan nilai fibonacci dengan formula elemen indeks ke-n-1 + elemen indeks ke-n-2. |

Pembuatan fungsi ini akan berguna untuk proses pemanggilan rekursif saat fungsi itu memanggil dirinya sendiri. Dengan begitu, pembuatan fungsi ini menjadi hal yang perlu diperhatikan dalam membuat algoritma rekursif.

### Pemanggilan Rekursif

Apakah program di atas berhenti sampai di situ? Program di atas masih kurang lengkap. Ketika nilai n sama dengan 0 atau 1, program di atas sudah benar adanya karena hanya terindikasi elemen 0 atau 1 saja.

Bagaimana dengan nilai n lebih dari 1? Program di atas tidak menyimpan nilai variabel apa pun. Ia hanya mencetak output berdasarkan nilai n. Di sinilah proses rekursif berlangsung. Program perlu memanggil fungsi itu sendiri untuk menghitung nilai dari elemen fibonacci.

Ingat dengan formula **fibonacci(5) = fibonacci(4) + fibonacci(3)**? Jika melihat formula tersebut, Anda memanggil fungsi indeks ke-4 dan ke-3 untuk mencari elemen indeks ke-5. Secara tidak langsung, ia akan menjumlahkan nilai fibonacci ke-4 dan ke-3.

Bagaimana dengan nilai fibonacci ke-4? Bagaimana cara menghitungnya? Dengan cara yang sama, ia akan memanggil fungsi fibonacci indeks ke-3 dan ke-2. Hal ini akan beruntun hingga mencapai indeks ke-1 dan ke-0. Jadi, program akan berubah menjadi seperti berikut.

|  |
| --- |
| 1. Tentukan elemen indeks **n** yang ingin dicari. 2. Buatlah fungsi bernama “fibonacci” dengan parameter **n**bertipe data integer positif.    1. Apabila nilai **n** sama dengan 0, silakan kembalikan nilai 0.    2. Apabila nilai **n** sama dengan 1, silakan kembalikan nilai 1.    3. Apabila nilai **n** lebih dari 1, hitung dan kembalikan nilai fibonacci dengan formula **fibonacci(n-1) + fibonacci(n-2)**. |

Ini adalah langkah ketika program mengulang fungsi untuk mencapai nilai **n** yang diharapkan.

### Base Case

Komponen terakhir adalah base case. Komponen ini adalah kondisi saat fungsi dapat mengembalikan output tanpa memanggil fungsi itu sendiri. Layaknya kondisi berhenti pada perulangan, base case ini akan menghentikan proses rekursi dan menampilkan output yang diharapkan.

Lalu, di manakah base case pada perhitungan fibonacci? Ya, betul sekali. Saat kita mendefinisikan output untuk n sama dengan 0 atau 1, program akan berhenti dan tidak akan mengulangi prosesnya lagi. Alhasil, keseluruhan program menjadi seperti berikut.

|  |
| --- |
| 1. Tentukan elemen indeks **n** yang ingin dicari. 2. Buatlah fungsi bernama “fibonacci” dengan parameter **n**bertipe data integer positif.    1. Apabila nilai **n** sama dengan 0, silakan **kembalikan nilai 0**.    2. Apabila nilai **n** sama dengan 1, silakan **kembalikan nilai 1**.    3. Apabila nilai **n** lebih dari 1, hitung dan kembalikan nilai fibonacci dengan formula fibonacci(n-1) + fibonacci(n-2). |

Apabila memanggil fungsi dengan argumen bernilai 0 atau 1, di situlah program itu berhenti. Ia tidak akan melanjutkan proses rekursi untuk kesekian kalinya.

Program telah selesai! Kita bisa mengonversi program tersebut menggunakan bahasa pemrograman Python. Alhasil, kode Python menjadi seperti berikut.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35123?from=35118#run1-editor1)

1

def fibonacci(n: int) -> int:

2

if n == 0:

3

return 0

4

elif n == 1:

5

return 1

6

else:

7

return fibonacci(n-1) + fibonacci(n-2)

8

​

9

n = 8

10

hitung\_fibonacci = fibonacci(n)

11

print(f"Fibonacci dari {n} adalah {hitung\_fibonacci}.")

12

​

13

# Output: Fibonacci dari 8 adalah 21.

 Input  Reset

 Jalankan

Materinya menarik sekali, bukan? Kita bisa menarik kesimpulan bahwa sebuah algoritma rekursif perlu memiliki empat komponen, yaitu proses pemecahan masalah, pembuatan fungsi, pemanggilan rekursif, dan base case. Komponen itu menjadi esensial untuk membuat algoritma rekursif. Algoritma rekursif tidak akan berjalan apabila salah satu komponen tersebut hilang.

Apabila Anda ingin mengeksplorasi algoritma rekursif atau fibonacci, silakan kunjungi beberapa tautan berikut.

* [Mastering recursive programming - IBM Developer](https://developer.ibm.com/articles/l-recurs/).
* [Recursion: a step-by-step introduction | by Isaac Wong | Medium](https://medium.com/@isaac_70614/recursion-a-step-by-step-introduction-ed25c957559c).
* [The Mathematical Magic of the Fibonacci Numbers](https://r-knott.surrey.ac.uk/Fibonacci/fibmaths.html).

## Fibonacci dalam Berbagai Aspek (Opsional)

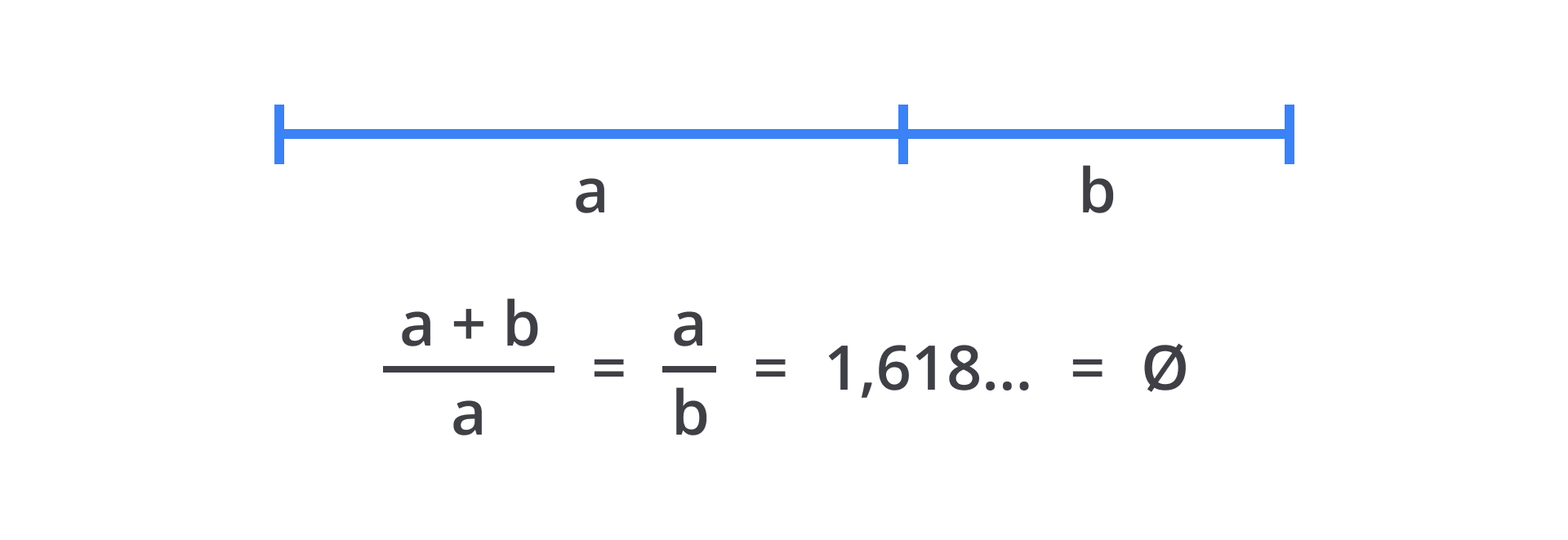
**Catatan:**

Materi ini bersifat opsional, yakni apabila ingin mempelajari fibonacci lebih dalam. Tidak ada keterkaitan antara materi ini dengan pemrograman. Dengan kata lain, Anda boleh menyimak serangkaian materi ini ataupun tidak.

Akan tetapi, materi ini sangatlah membantu Anda dalam membuka wawasan berpikir mengenai fibonacci. Jadi, perhatikan materi ini dengan saksama, ya.

Pada materi sebelumnya, kita membahas mengenai komponen esensial yang harus ada pada konsep rekursi. Kita mengulik komponen tersebut dari salah satu penerapan rekursi, yaitu deret fibonacci.

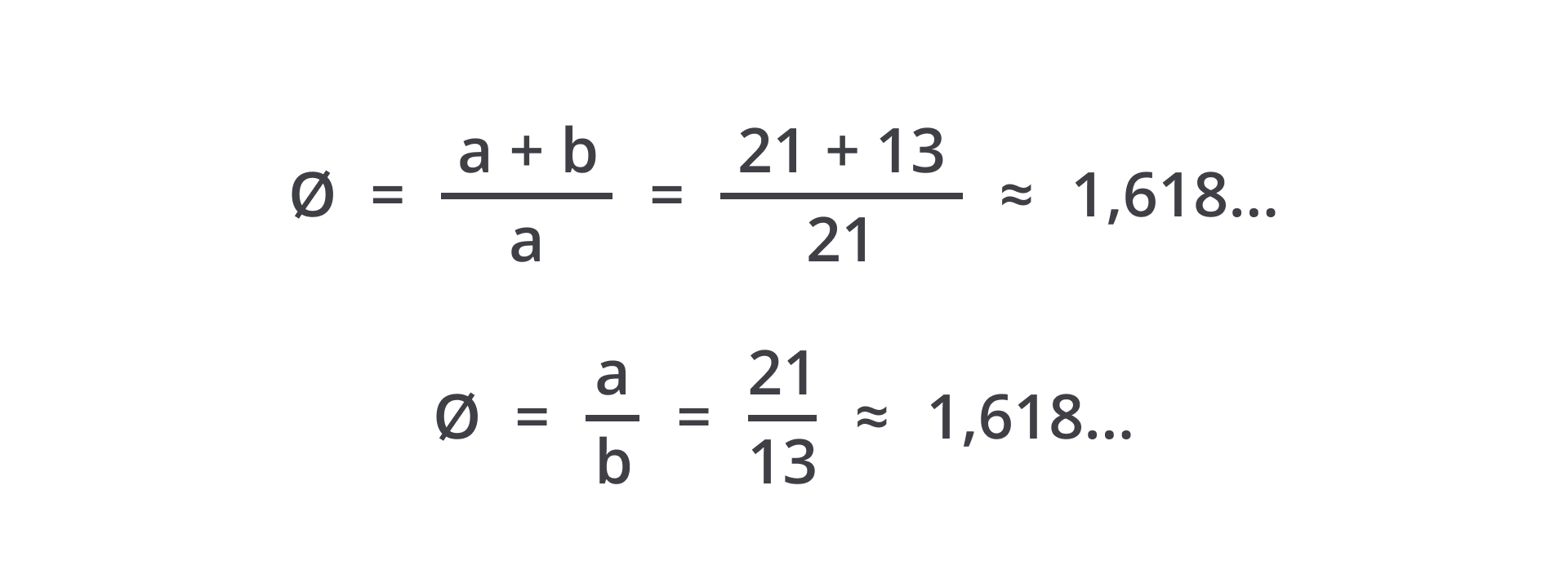
Deret fibonacci ini memiliki keunikan tersendiri. Ia memiliki proporsi unik antara dua angka yang berurutan. Dalam dunia matematis, proporsi tersebut disebut sebagai **golden ratio** (rasio emas). Rasio ini diberi simbol Yunani bernama “phi” ([](https://www.dicoding.com/academies/620/tutorials/35128?from=35123)). Rasio emas didapatkan dengan perhitungan seperti berikut.

[](https://www.dicoding.com/academies/620/tutorials/35128?from=35123)

Berikut adalah penjelasan dari persamaan di atas.

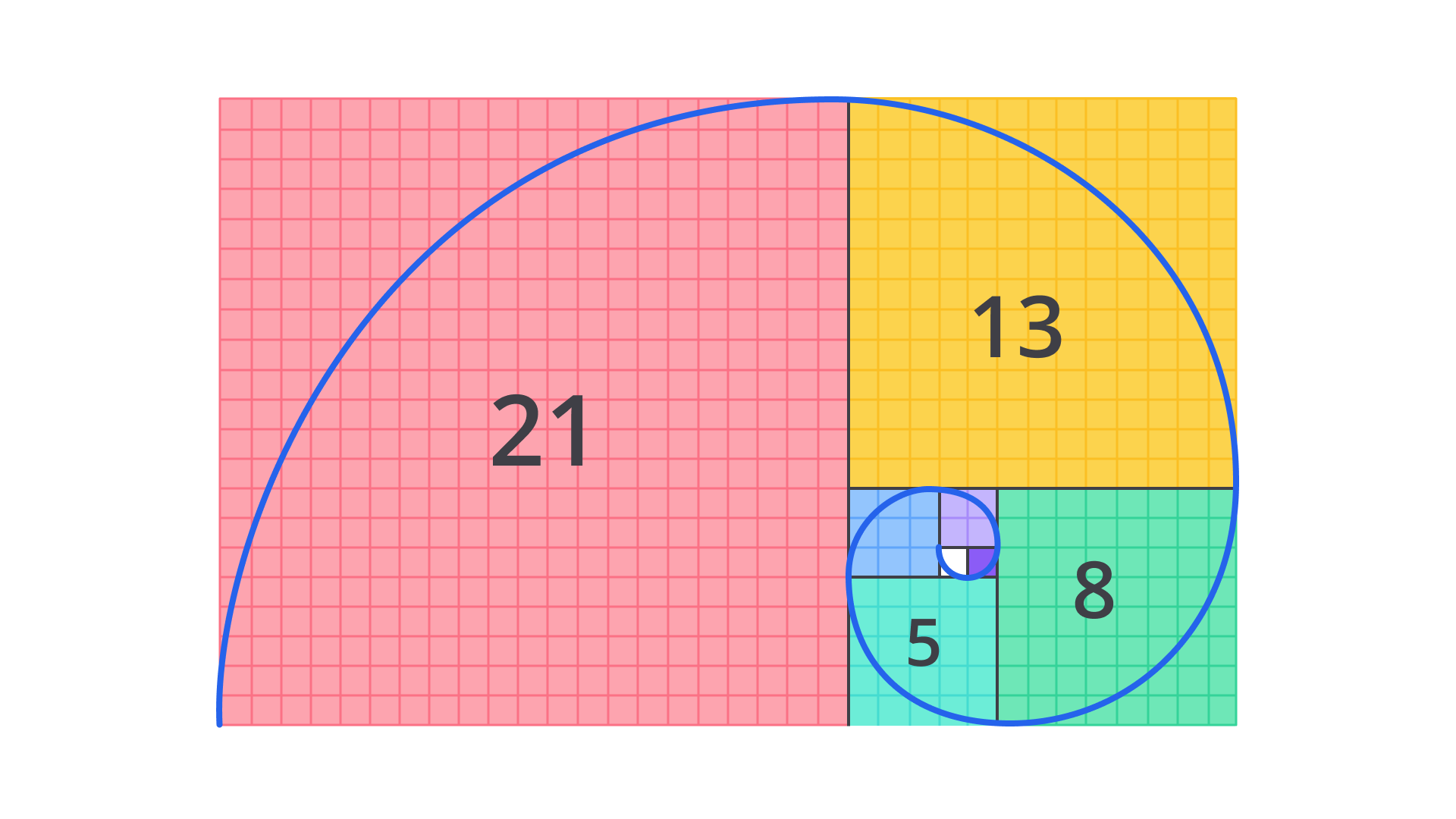
* **a** adalah suatu bilangan deret fibonacci pada indeks ke-n.
* **b**adalah suatu bilangan deret fibonacci pada indeks ke-n-1 atau sebelum nilai a.
* adalah golden ratio bernilai 1.618 sekian.

Mari kita menghitung salah satu nilai dari elemen fibonacci. Misalnya, jika **a** bernilai 21, **b** bernilai 13. Ingat, kedua nilai harus bersebelahan. Nilai 21 dan 13 tepat bersebelahan sehingga kita bisa mencoba menghitung golden ratio. Alhasil, perhitungannya akan seperti berikut.

[](https://www.dicoding.com/academies/620/tutorials/35128?from=35123)

Nilai dari kedua formula tersebut mendekati golden ratio, yaitu 1,618. Anda dapat memilih angka berapa pun pada deret fibonacci. Apabila dihitung dengan formula tersebut, output-nya selalu mendekati 1,618. Hal ini yang menjadikan nilai tersebut sebagai golden ratio.

Masih ingatkah Anda dengan pola spiral pada deret fibonacci? Iya, pola yang membentuk seperti pusaran angin itu adalah hasil kombinasi pembuatan persegi dengan golden ratio. Hal ini merujuk pada perbandingan panjang sisi antara persegi satu dengan lainnya. Jadi, menghasilkan golden rectangle seperti berikut.

[](https://www.dicoding.com/academies/620/tutorials/35128?from=35123)

Selain itu, golden ratio ini menyimpan banyak rahasia. Anda dapat menemukan golden ratio atau deret fibonacci dalam berbagai aspek. Mau tahu apa rahasianya? Simak tuntas materinya, ya.

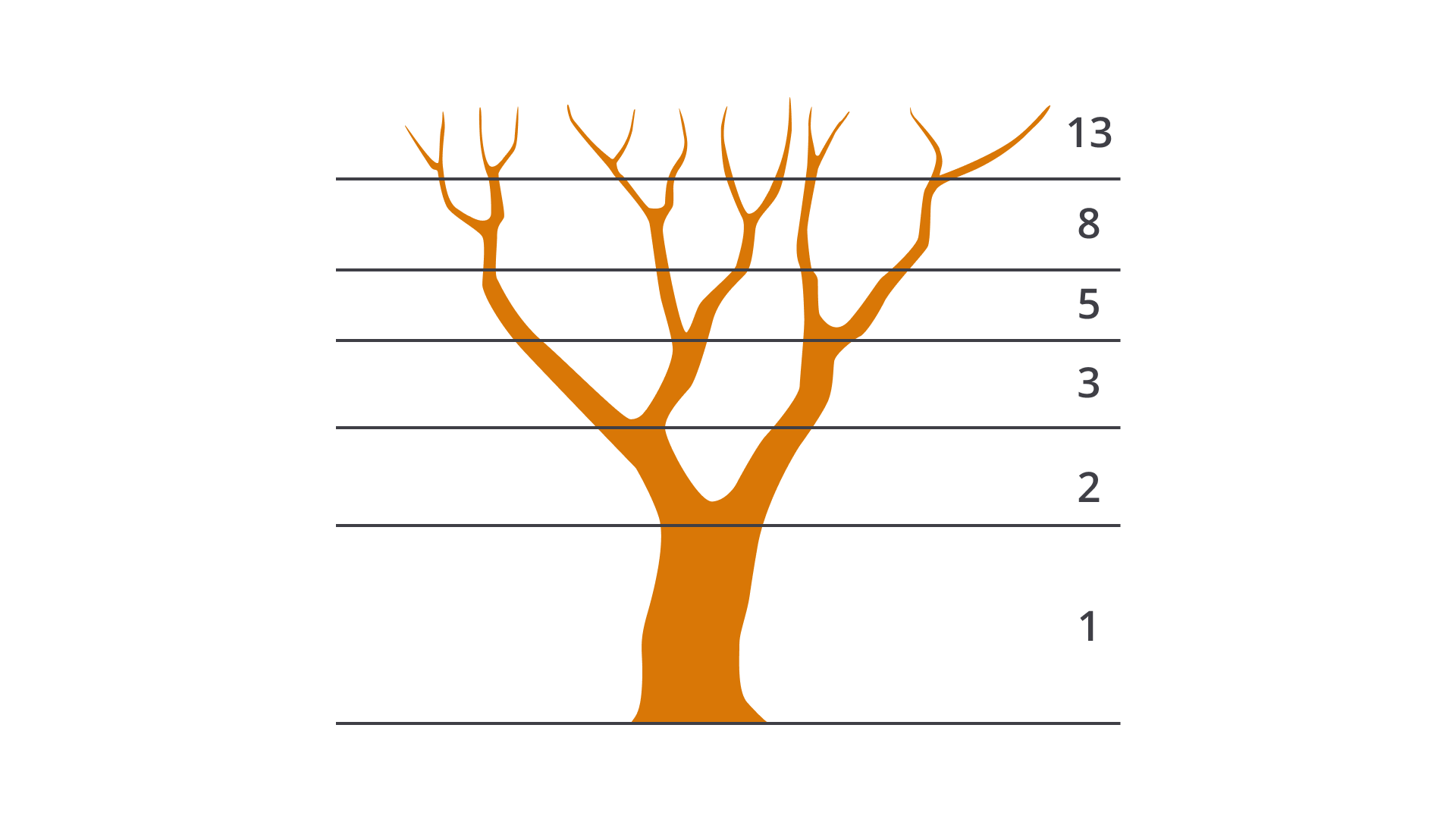
### Flora

Pernahkah Anda melihat pola golden ratio pada tumbuhan atau tanaman? Mungkin Anda tidak sadar telah melihat pola tersebut. Beberapa tumbuhan memiliki daun, bunga, bahkan batang yang mengikuti golden ratio. Maksudnya adalah perbandingan ukuran, posisi, atau jumlah dari struktur tumbuhan mengikuti golden ratio.

#### Batang Pohon

Pertama adalah batang pohon. Ternyata, deret fibonacci juga ada di pohon, lo. Perhatikan batang pohon tumbuh dari atas permukaan. Ia akan tumbuh dengan satu batang yang besar. Kemudian, ia akan bercabang menjadi dua setelah kurun waktu tertentu. Lalu, ia akan tumbuh bercabang hingga pohon tersebut rimbun.

Jika diamati, jumlah batang pohon tersebut mengikuti perbandingan golden ratio dengan dimulai dari satu batang, bercabang menjadi dua, dan seterusnya. Anda dapat melihat ilustrasi berikut agar dapat membayangkannya dengan mudah.

[](https://www.dicoding.com/academies/620/tutorials/35128?from=35123)

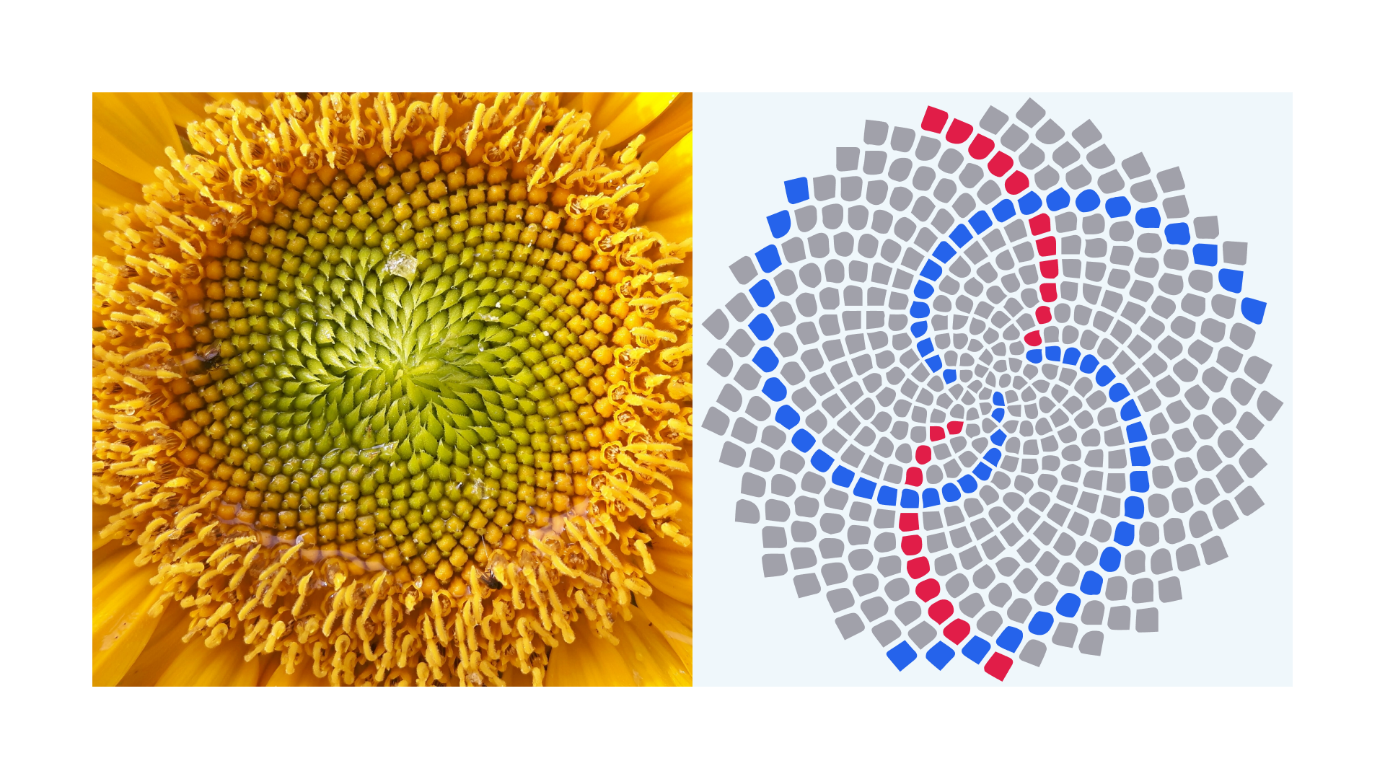
#### Daun Tumbuhan

Pernahkah Anda melihat daun atau kelopak bunga berjumlah layaknya deret fibonacci? Ada tanaman yang memiliki daun berjumlah seperti deret fibonacci. Biasanya, tanaman sukulen mempunyai layer daun yang jumlahnya menyerupai deret fibonacci. Selain itu, arah perputaran daun bertumbuh pun mengikuti pola golden rectangle. Anda dapat melihat pola fibonacci tumbuhan pada gambar berikut.

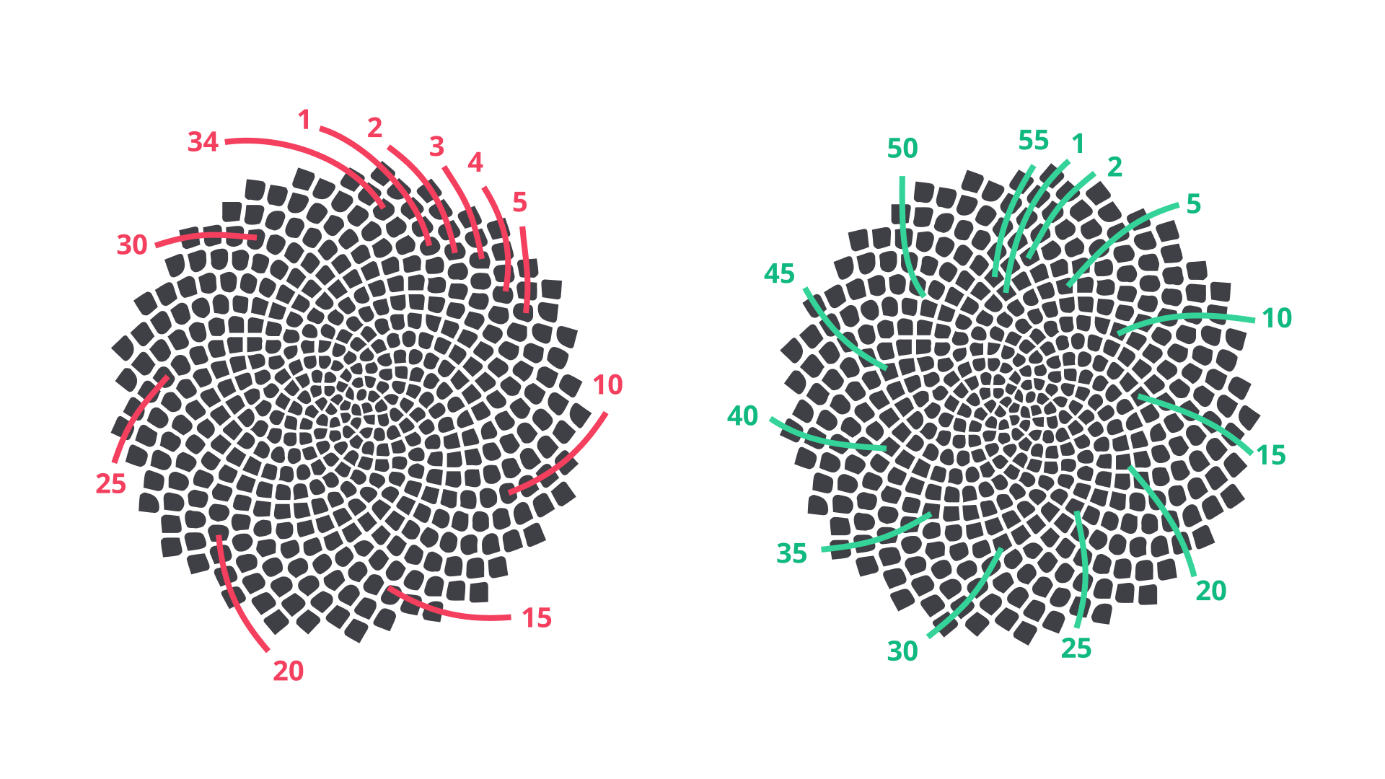
[](https://www.dicoding.com/academies/620/tutorials/35128?from=35123)

#### Bunga Matahari

Pernahkah melihat bunga matahari? Lalu, apakah Anda pernah mengamati ada elemen fibonacci pada bunga tersebut? Perhatikanlah bunga matahari saat sedang mekar. Cantik sekali, bukan? Di balik itu, ternyata ada dua jenis spiral unik pada benih di tengah kepala bunga. Jika ditelusuri, ada spiral yang memiliki pola clockwise (searah jarum jam) dan counter-clockwise (berlawanan arah jarum jam). Anda dapat melihatnya pada ilustrasi bunga matahari berikut.

[](https://www.dicoding.com/academies/620/tutorials/35128?from=35123)

Menurut Profesor Fisika H.E. Huntley [7], jumlah spiral antara clockwise dan counter-clockwise berbeda. Ada yang menyebutkan bahwa spiral pada pola clockwise berjumlah 34, sedangkan counter-clockwise berjumlah 55. Kedua angka tersebut merupakan elemen dari deret fibonacci yang bersebelahan.

[](https://www.dicoding.com/academies/620/tutorials/35128?from=35123)

### Fauna

Tidak hanya flora, nyatanya ada pola deret fibonacci pada fauna atau hewan juga. Bentuk spiral dari pola deret fibonacci dapat Anda temukan pada beberapa hewan, seperti keong, siput, dan shell atau kerang. Pusat lingkaran hingga ujung cangkang membentuk sebuah spiral layaknya spiral fibonacci. Ukuran perputaran spiral memiliki golden ratio dan berulang hingga ujung.

[](https://www.dicoding.com/academies/620/tutorials/35128?from=35123)

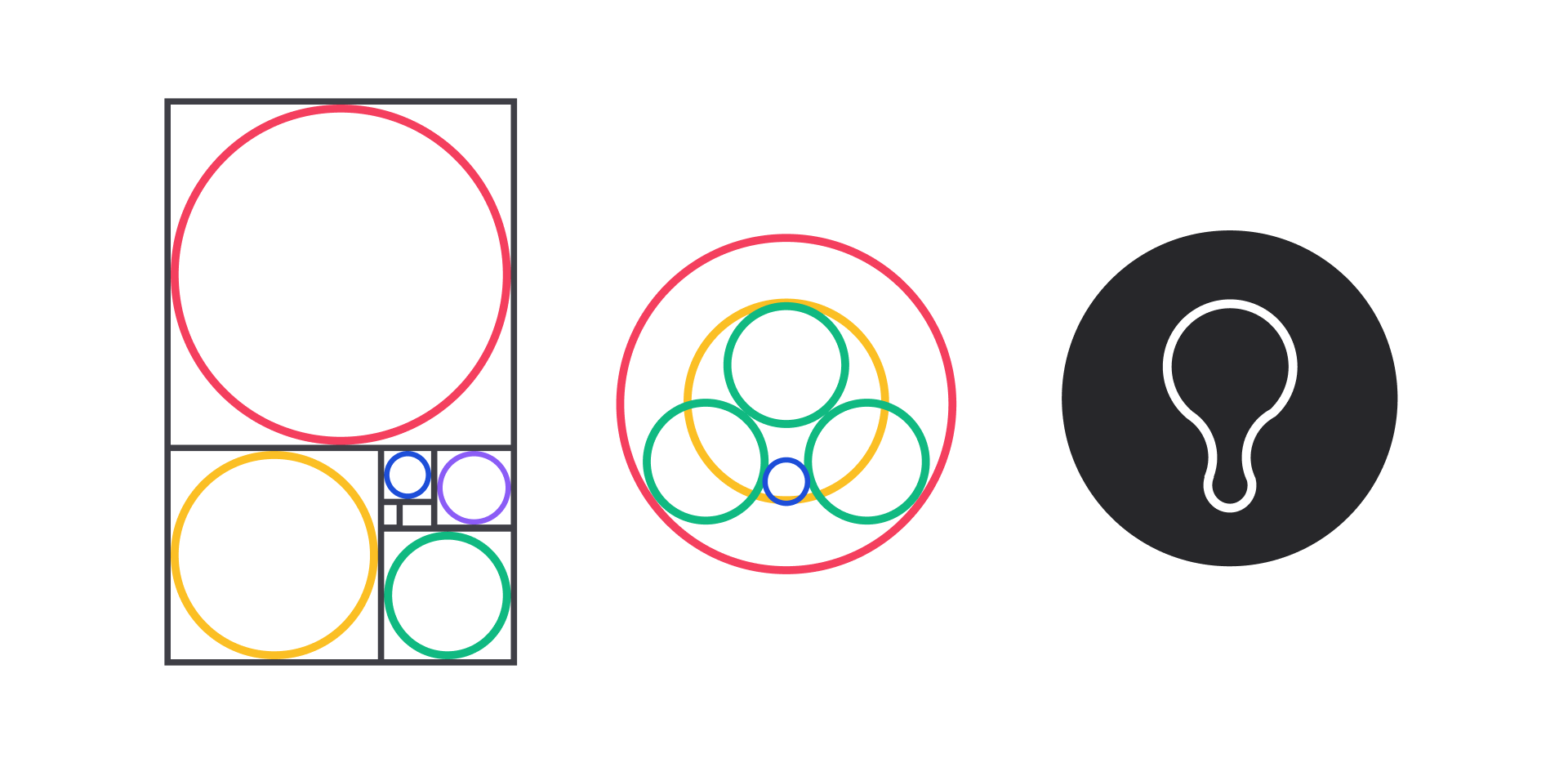
Menariknya lagi, deret fibonacci juga muncul dalam kondisi yang tidak Anda duga. Anda tahu bunglon? Ya, hewan yang mampu berkamuflase pada lingkungannya. Coba Anda perhatikan ekor dari bunglon. Ia melingkar dan berpusat di tengah sehingga menyerupai spiral fibonacci.

[](https://www.dicoding.com/academies/620/tutorials/35128?from=35123)

### Desain

Dunia desain juga menerapkan konsep fibonacci ini. Seorang desainer akan lebih mudah membuat ilustrasi dengan outline yang dibangun berdasarkan bentuk dasar, seperti bujur sangkar dan lingkaran. Dengan bentuk dasar tersebut, ilustrasi atau gambar akan semakin estetik, memiliki porsi yang tepat, dan seimbang antara garis satu dengan lainnya.

Ada konsep fibonacci yang dapat diterapkan dalam dunia desain, yaitu golden ratio. Ia memanfaatkan ukuran elemen setiap bentuk dengan besaran sesuai rasio 1:1,618. Dengan begitu, desain akan memiliki perpaduan proporsi yang estetis sehingga menciptakan kesan visual yang menarik dan alami.

[](https://www.dicoding.com/academies/620/tutorials/35128?from=35123)

### Alam

Selain tumbuhan dan hewan, ternyata alam pun memiliki pola layaknya spiral fibonacci. Pastinya Anda mengenal galaksi bima sakti. Iya, galaksi tempat bumi kita berada. Ia tersusun atas bintang, planet, gas, debu, dan masih banyak lainnya. Benda-benda tersebut beterbangan dan berotasi mengelilingi inti galaksi. Pergerakan galaksi bima sakti ini dapat diibaratkan seperti planet-planet yang mengelilingi matahari dalam tata surya.

Jika diperhatikan, perputaran benda-benda pada galaksi tersebut menyerupai spiral fibonacci. Gas, debu, planet, dan lainnya seolah berputar pada inti galaksi dan membentuk spiral. Agar memudahkan Anda membayangkan perputaran benda galaksi, silakan melihat ilustrasi berikut.

[](https://www.dicoding.com/academies/620/tutorials/35128?from=35123)

Mungkin galaksi terlalu jauh untuk dilihat oleh mata kita. Saat pergi ke pantai, Anda pasti melihat laut. Deburan ombak memecah kesunyian membuat kita nyaman saat mendengar dan melihatnya. Namun, coba Anda perhatikan dengan saksama. Gelombang ombak yang Anda lihat itu menyerupai spiral fibonacci.

[](https://www.dicoding.com/academies/620/tutorials/35128?from=35123)

Ternyata, deret fibonacci atau golden ratio berada di sekitar kita, ya. Deret fibonacci yang dihasilkan dengan menjumlahkan dua angka sebelumnya itu sering kita temukan dalam pola-pola alam dan struktur. Masih banyak aspek lainnya yang mengandung unsur deret fibonacci atau golden ratio, seperti bentuk telinga, arsitektur gedung, komposisi musik, dll.

Apabila Anda ingin mengeksplorasi fibonacci, silakan mengulik beberapa tautan berikut, ya.

* [Fibonacci and Lucas Numbers with Applications, Volume 1, 2nd Edition](https://learning.oreilly.com/library/view/fibonacci-and-lucas/9781118742129/).
* [Golden Ratio. Bring Balance in UI Design. | by tubik](https://uxplanet.org/golden-ratio-bring-balance-in-ui-design-765c954f0ff9)
* [What is the Golden Ratio?](https://medium.com/i-math/what-is-the-golden-ratio-d3cc17c8fefd)

## Jenis Algoritma Rekursif

Sebagaimana materi sebelumnya, cara kerja algoritma rekursif ini menyelesaikan permasalahan yang besar dengan membaginya menjadi potongan masalah kecil. Ia dapat diselesaikan dengan menerapkan solusi yang serupa. Dengan begitu, penyelesaian masalah dapat dibungkus dengan sebuah fungsi yang memanggil fungsi itu sendiri.

Namun, tahukah Anda bahwa algoritma rekursif memiliki beberapa jenis untuk menyelesaikan permasalahan? Ya, algoritma rekursif memiliki tiga cara unik untuk menyelesaikan permasalahan, yaitu berdasarkan hal berikut:

* struktur pemanggilan fungsi;
* metode pemanggilan fungsi; dan
* eksekusi pemanggilan fungsi.

Anda sudah mulai penasaran, bukan? Yuk, simak perbedaan ketiga cara tersebut hingga tuntas.

### Struktur Pemanggilan Fungsi

Salah satu hal mendasar dalam algoritma rekursif adalah pemanggilan fungsi yang berulang-ulang. Kita perlu mengetahui seberapa kompleks fungsi yang akan dipanggil. Jadi, kita dapat menyusunnya secara terstruktur dan terorganisasi dengan baik.

Pada algoritma rekursif, ada dua cara dalam menyelesaikan permasalahan berdasarkan struktur pemanggilan fungsi, yakni linear recursion dan tree recursion.

#### Linear Recursion

Linear recursion adalah suatu fungsi rekursif yang hanya memanggil satu kali fungsi itu sendiri dalam setiap pemanggilannya. Coba perhatikan kode berikut.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35133?from=35128#run1-editor1)

1

def counting\_number(n : int):

2

if n > 0:

3

print(f"{n}", end=" ")

4

counting\_number(n-1)

5

​

6

counting\_number(5)

7

print()

8

​

9

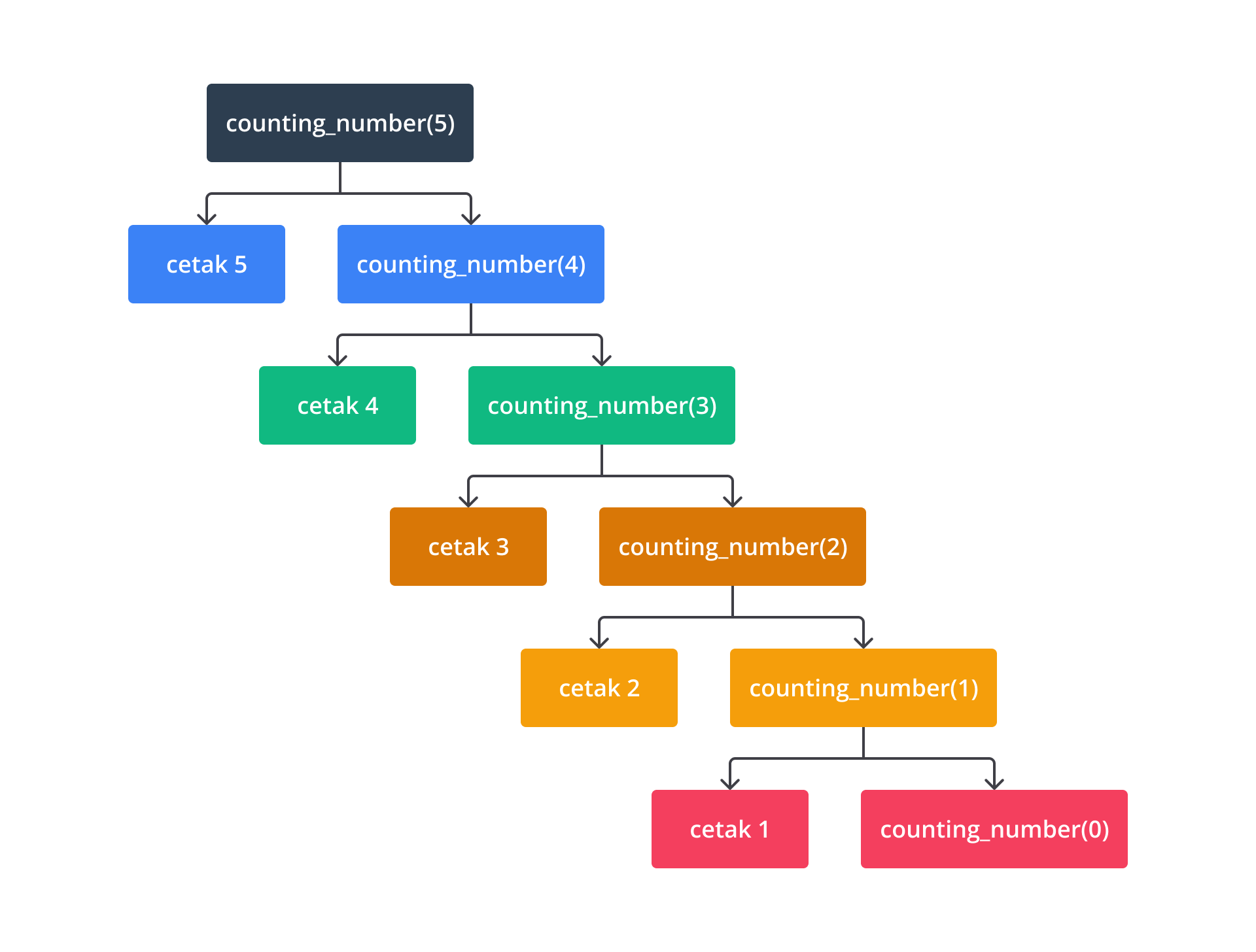
# Output: 5 4 3 2 1

 Input  Reset

 Jalankan

Kode di atas adalah contoh algoritma rekursif yang bertujuan untuk menampilkan angka berurutan mulai dari nilai n hingga 1. Ia memanggil fungsi counting\_number untuk mencetak angka sesuai yang diminta.

Namun, coba lihat isi dari fungsi tersebut. Ia memiliki satu aksi memanggil fungsi itu sendiri dan tidak ada aksi rekursif lainnya. Apabila diurai dan ditampilkan dalam pohon pemanggilan fungsi, ia akan menjadi seperti berikut.

[](https://www.dicoding.com/academies/620/tutorials/35133?from=35128)

Pohon pemanggilan fungsi hanya terbagi menjadi dua aksi, yaitu mencetak angka dan memanggil fungsi. Hal itu berulang hingga kondisi base case terpenuhi. Inilah yang disebut sebagai linear recursion.

#### Tree Recursion

Berbeda dengan linear recursion, tree recursion terjadi ketika fungsi rekursif memanggil fungsi itu sendiri lebih dari satu kali. Perhatikan algoritma rekursif fibonacci berikut.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35133?from=35128#run2-editor1)

1

def fibonacci(n: int) -> int:

2

if n == 0:

3

return 0

4

elif n == 1:

5

return 1

6

else:

7

return fibonacci(n-1) + fibonacci(n-2)

8

​

9

n = 5

10

hitung\_fibonacci = fibonacci(n)

11

print(f"Fibonacci dari {n} adalah {hitung\_fibonacci}.")

12

​

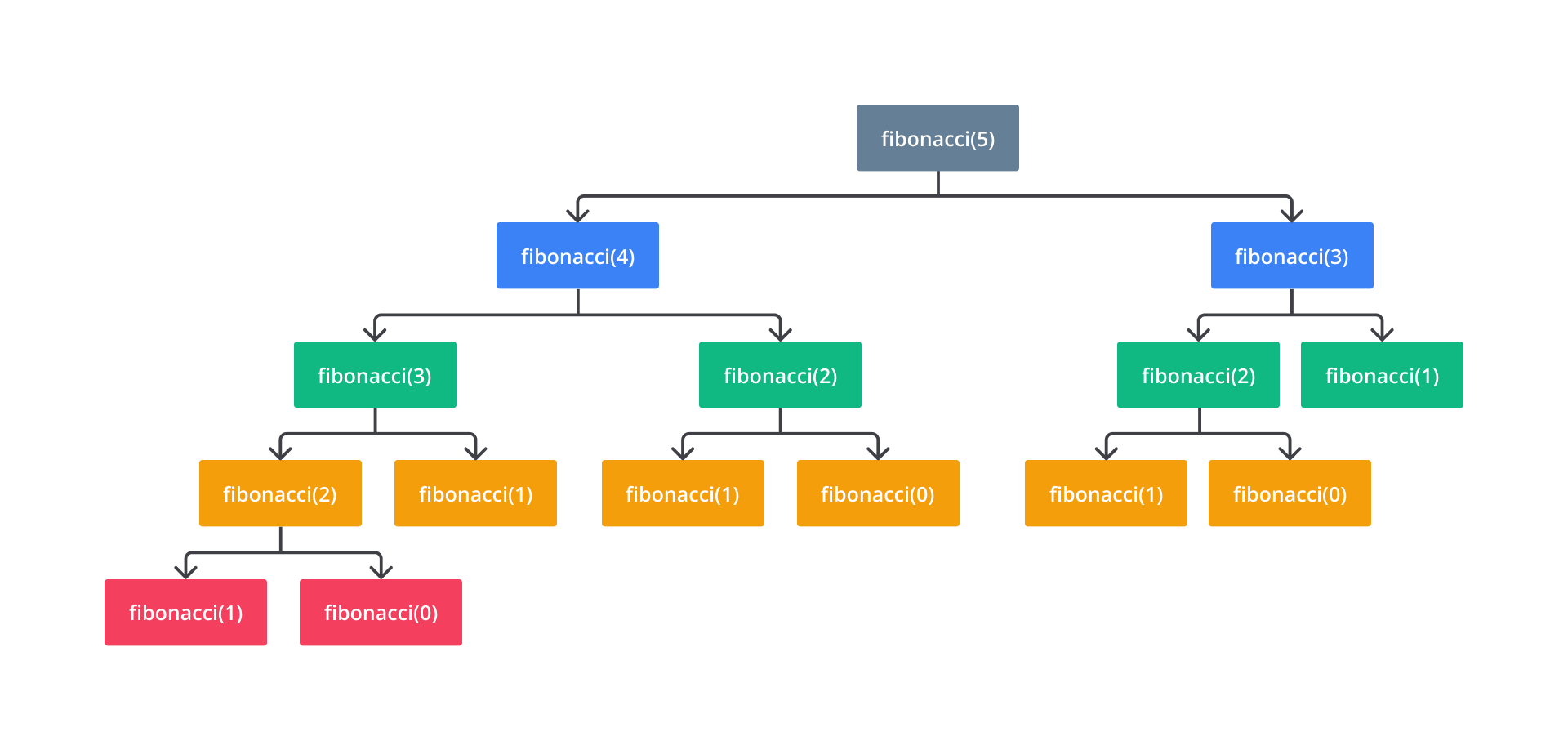
13

# Output: Fibonacci dari 5 adalah 5.

 Input  Reset

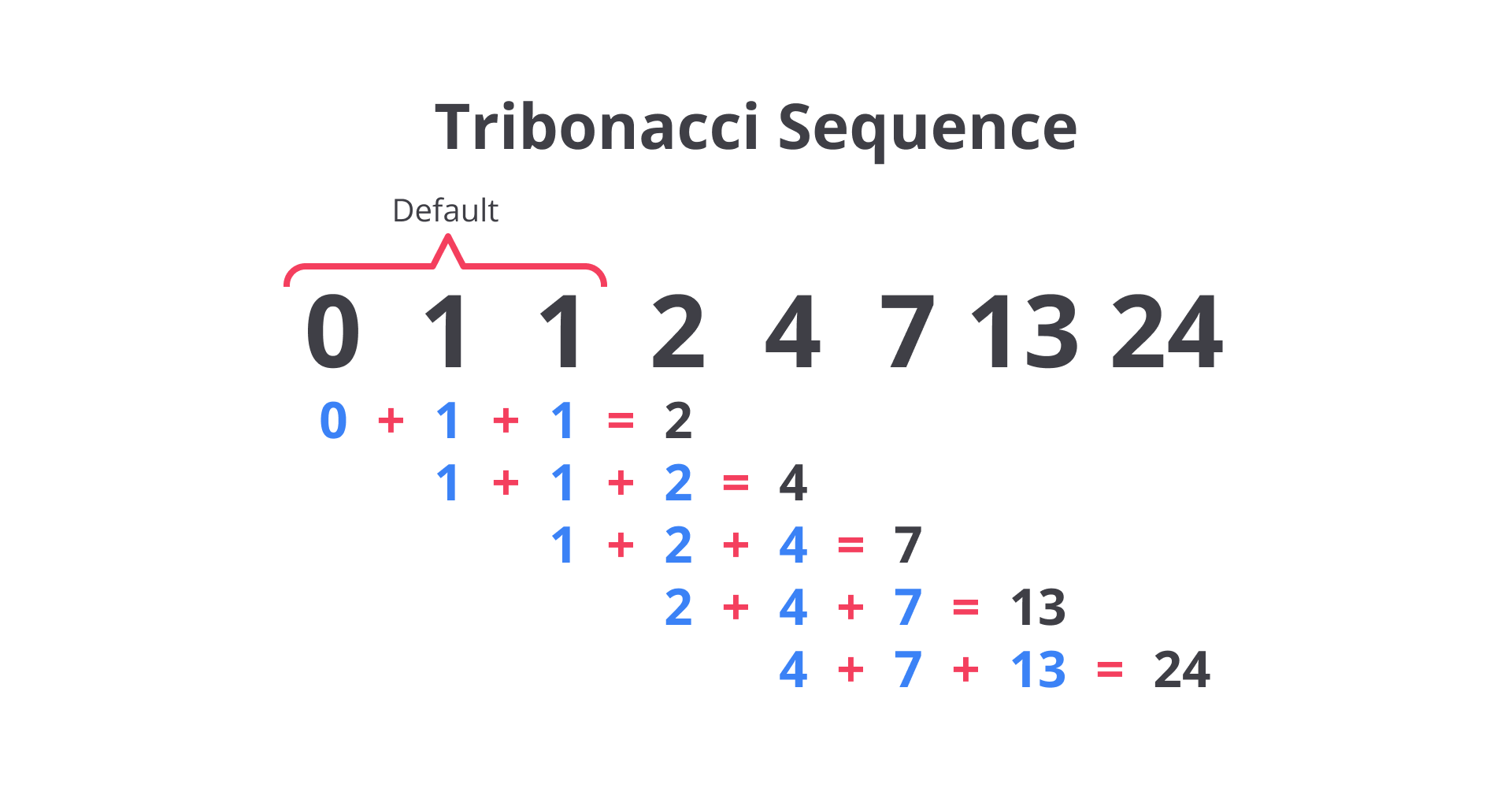
 Jalankan

Pada nilai n sama dengan 0 atau 1, fungsi berada dalam kondisi base case atau tidak memanggil fungsi rekursif. Ketika nilai n lebih dari 1, ia memanggil fungsi rekursif sebanyak dua kali. Jika disusun pada pohon pemanggilan fungsi, ia akan menjadi seperti berikut.

[](https://www.dicoding.com/academies/620/tutorials/35133?from=35128)

Perhatikan bagan di atas. Setiap fungsi rekursif berjalan, program akan memanggil dua fungsi sekaligus. Tujuannya adalah untuk mengetahui dua angka sebelum nilai yang diinginkan. Alhasil, program fibonacci membutuhkan dua pemanggilan sekaligus untuk mengetahui angka tersebut. Hal ini yang menyebabkan fibonacci masuk dalam kategori tree recursion.

Selain fibonacci, ada deret bilangan [tribonacci](https://mathworld.wolfram.com/TribonacciNumber.html). Deret tersebut mirip seperti fibonacci, tetapi dihitung dengan menjumlahkan tiga bilangan sebelumnya. Untuk fibonacci, bilangan yang sudah ditetapkan hanya elemen 0 dan 1. Akan tetapi, deret tribonacci memiliki tiga bilangan yang tetap, yaitu tribo[0] = 0, tribo[1] = 1, dan tribo[2] = 1. Jadi, deret tribonacci akan seperti berikut.

[](https://www.dicoding.com/academies/620/tutorials/35133?from=35128)

Untuk menghitung elemen tribonacci, Anda hanya mengubah konsep program fibonacci menjadi seperti berikut.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35133?from=35128#run3-editor1)

1

def tribonacci(n: int) -> int:

2

if n == 0:

3

return 0

4

elif n == 1 or n == 2:

5

return 1

6

else:

7

return tribonacci(n-1) + tribonacci(n-2) + tribonacci(n-3)

8

​

9

​

10

n = 4

11

hitung\_tribonacci = tribonacci(n)

12

print(f"Deret tribonacci ke {n} adalah {hitung\_tribonacci}.")

13

​

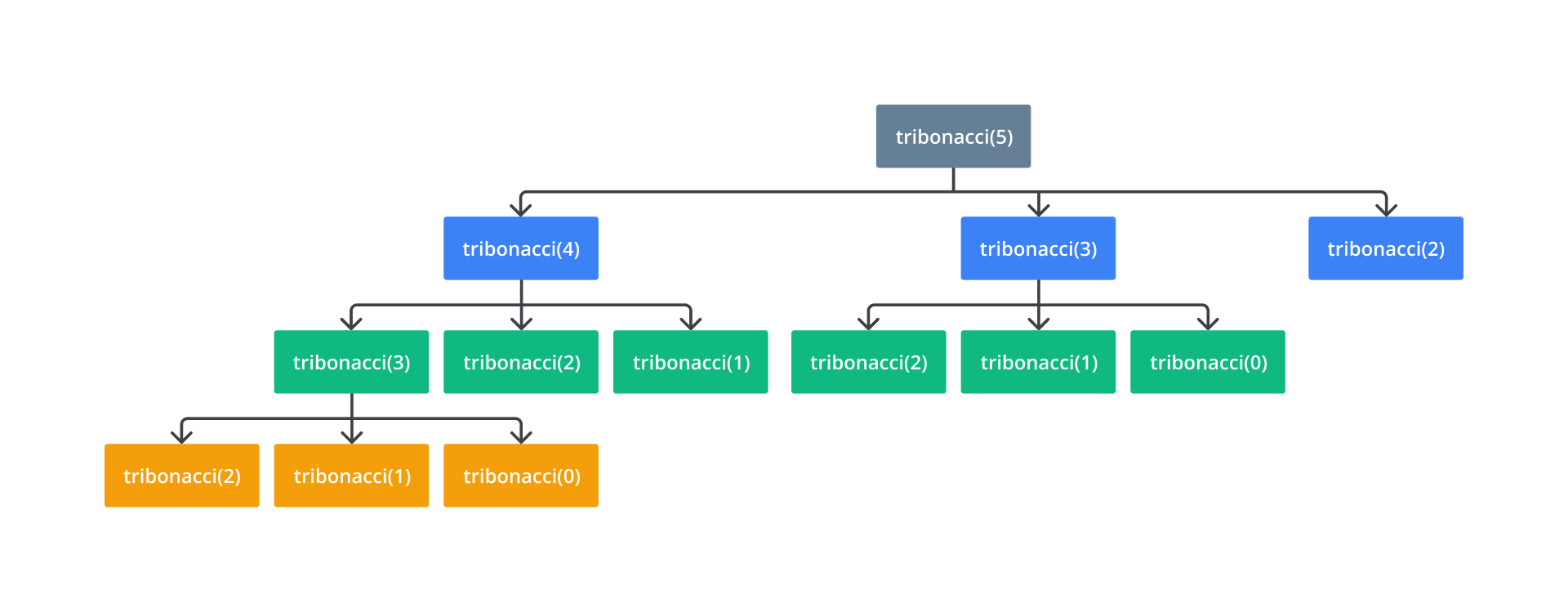
14

# Output: Deret tribonacci ke 5 adalah 7.

 Input  Reset

 Jalankan

Melihat program di atas, fungsi tribonacci() dipanggil sebanyak tiga kali dalam satu fungsi. Apabila ditampilkan dalam pohon pemanggilan fungsi, strukturnya akan seperti berikut.

[](https://www.dicoding.com/academies/620/tutorials/35133?from=35128)

Berdasarkan struktur di atas, pemanggilan fungsi tribonacci akan memanggil fungsi rekursif sebanyak tiga kali. Hal tersebut bertujuan agar dapat menghitung bilangan deret pada elemen berikutnya. Pemanggilan fungsi akan berakhir ketika elemen bernilai 0, 1, atau 2 karena ketiga nilai tersebut merupakan base case. Dengan konsep pemanggilan tersebut, tribonacci memiliki struktur yang menyerupai tree atau pohon dan termasuk dalam kategori tree recursion.

### Metode Pemanggilan Fungsi

Sebagaimana cara sebelumnya, pemanggilan fungsi yang berulang-ulang menjadi aspek yang perlu diperhatikan pada algoritma rekursif. Hal ini memiliki dampak signifikan terhadap kinerja dan efisiensi algoritma yang kita implementasikan. Jangan sampai pemanggilan fungsi dapat merusak kinerja algoritma dalam menyelesaikan permasalahan.

Nyatanya, ada dua metode dalam memanggil fungsi pada algoritma rekursif, yakni direct recursion dan indirect recursion.

#### Direct Recursion

Berdasarkan namanya, direct recursion merupakan konsep rekursif yang memanggil fungsi secara langsung. Tidak ada perantara untuk mengulang proses pemanggilan fungsi. Jadi, isi dari fungsi rekursif memiliki aksi memanggil fungsi itu sendiri. Coba perhatikan kode berikut.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35133?from=35128#run4-editor1)

1

def counting\_number(n : int):

2

if n > 0:

3

print(f"{n}", end=" ")

4

counting\_number(n-1)

5

​

6

counting\_number(5)

7

print()

8

​

9

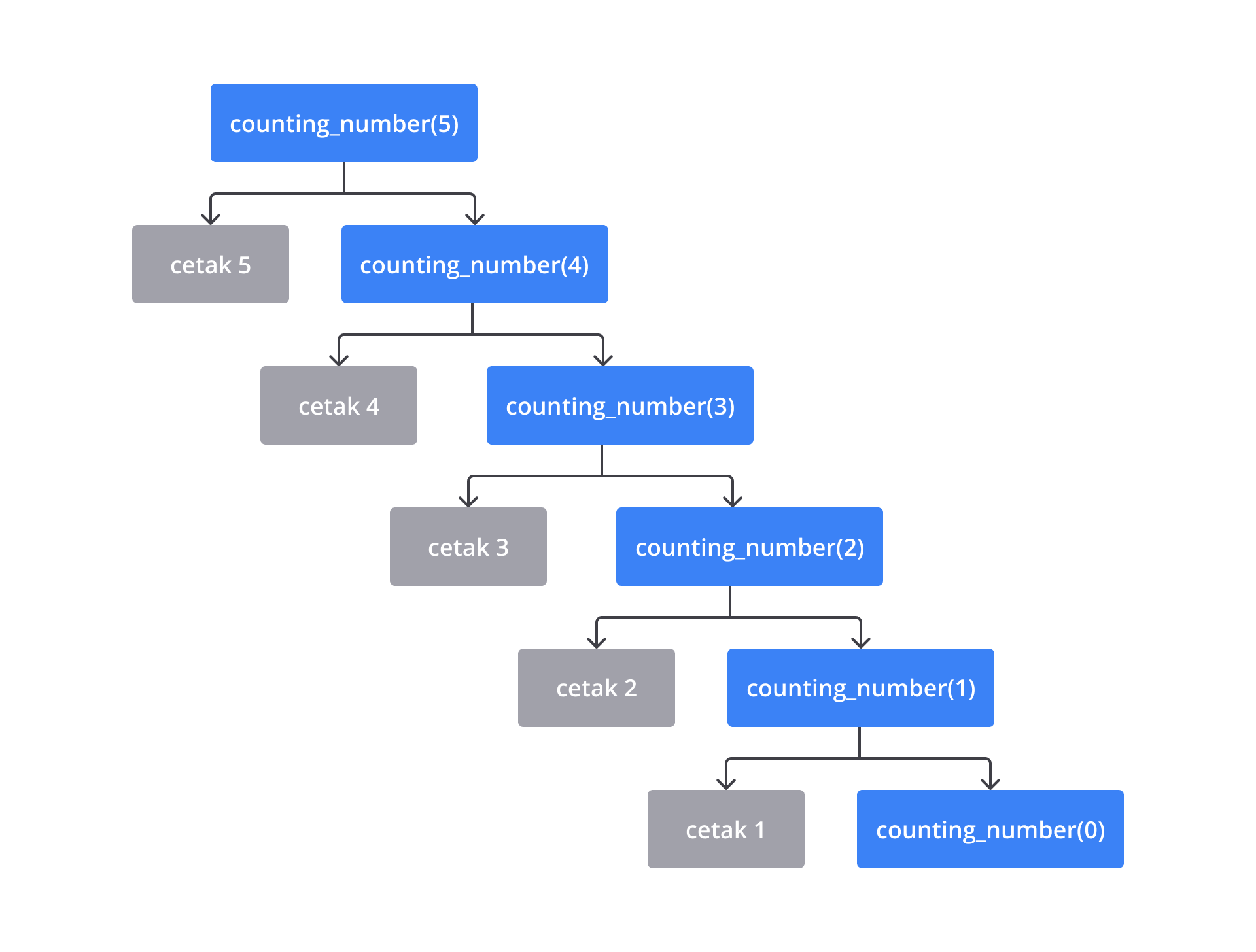
# Output: 5 4 3 2 1

 Input  Reset

 Jalankan

Kode di atas adalah contoh algoritma rekursif untuk mencetak angka berurutan, mulai dari nilai n hingga 1. Fungsi rekursif yang dipanggil pertama kali bernama counting\_number.

Mengapa fungsi tersebut dapat dikatakan sebagai contoh algoritma rekursif? Iya, fungsi **counting\_number** berisi aksi memanggil dirinya sendiri. Ia secara langsung memanggil fungsi di akhir blok program. Coba perhatikan bagan berikut supaya Anda lebih paham maksud dari direct recursion.

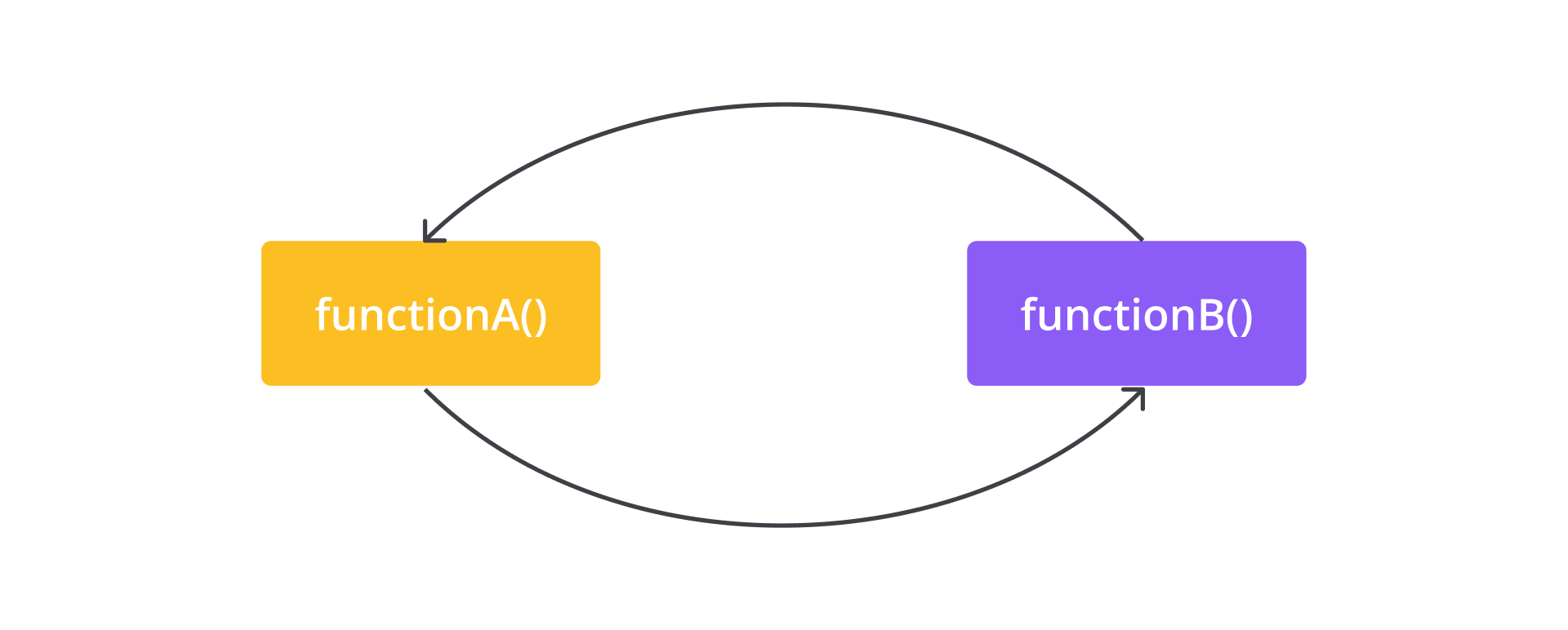
[](https://www.dicoding.com/academies/620/tutorials/35133?from=35128)

Berdasarkan ilustrasi di atas, setiap fungsi **counting\_number** memiliki aksi, yaitu memanggil fungsi itu sendiri. Ia tidak menjalankan fungsi selain counting\_number sebagai perantara. Hal ini berlaku pada program rekursif fibonacci yang memanggil fungsi dirinya sendiri.

#### Indirect Recursion

Berbeda dengan direct recursion, indirect recursion adalah jenis rekursi yang menjalankan fungsi rekursif secara tidak langsung. Ia membutuhkan perantara untuk mengulang fungsi rekursif. Ia menggunakan fungsi tambahan agar fungsi rekursif dijalankan kembali.

Anda dapat membayangkan bahwa fungsi A akan memanggil fungsi B. Lalu di dalamnya akan memanggil fungsi A atau fungsi lain yang berakhir pada fungsi A. Jadi, ia akan membentuk sebuah siklus pemanggilan fungsi yang berakhir pada fungsi rekursif awal.

[](https://www.dicoding.com/academies/620/tutorials/35133?from=35128)

Coba perhatikan kode berikut.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35133?from=35128#run5-editor1)

1

def isOdd(number: int):

2

if number == 0:

3

return False

4

else:

5

return isEven(number-1)

6

​

7

def isEven(number: int):

8

if number == 0:

9

return True

10

else:

11

return isOdd(number-1)

12

13

print(isOdd(5))

14

​

15

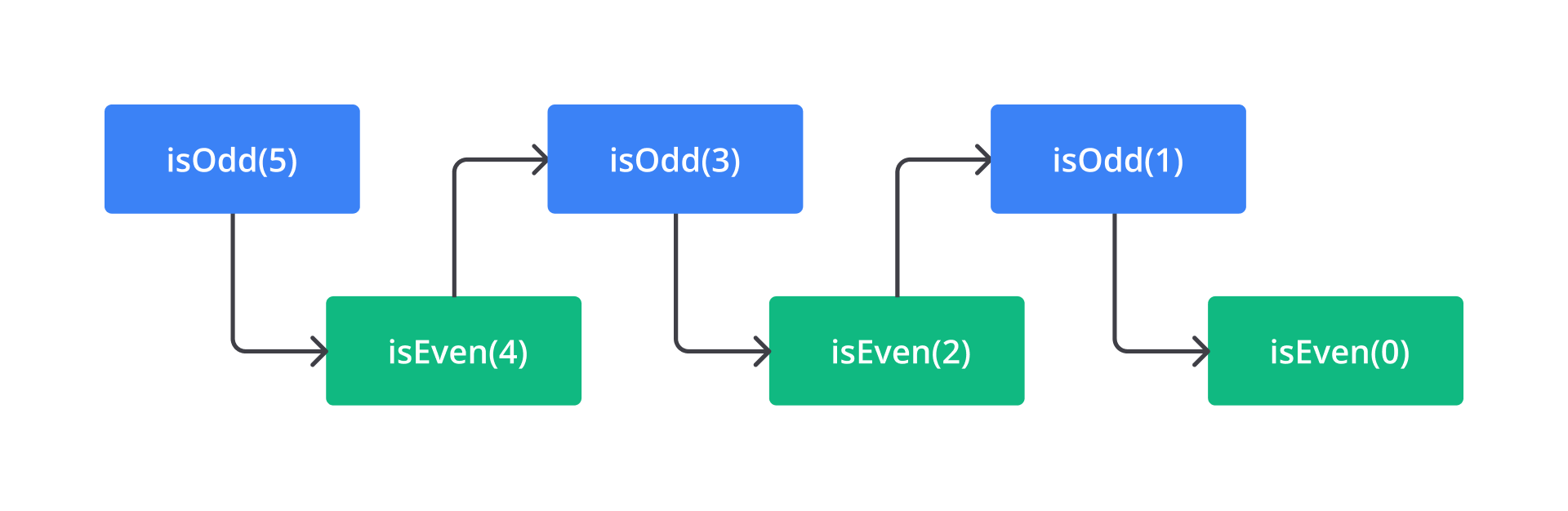
# Output: True

 Input  Reset

 Jalankan

Kode di atas adalah contoh algoritma rekursif yang bertujuan untuk mengetahui angka genap atau ganjil. Konsep kerjanya dengan mengurangi satu dari angka yang diinginkan hingga mencapai angka nol. Apabila angka nol ditemukan pada fungsi isOdd, program akan mengembalikan nilai **False**. Sebaliknya, program akan mengembalikan nilai **True** jika angka nol ditemukan pada fungsi isEven.

Perhatikan kembali fungsi pada kode di atas. Isi dari fungsi isOdd bukanlah memanggil dirinya sendiri, melainkan memanggil fungsi isEven. Akan tetapi, isi dari fungsi isEven memanggil isOdd. Program akan berulang pada siklus isOdd-isEven-isOdd hingga number bernilai nol.

[](https://www.dicoding.com/academies/620/tutorials/35133?from=35128)

Konsep inilah yang disebut sebagai indirect recursion, yakni konsep rekursif dengan penambahan fungsi baru sebagai perantara untuk berulang. Ia tidak serta-merta memanggil fungsi diri sendiri, tetapi butuh fungsi tambahan untuk menjalankan aksi rekursif.

### Eksekusi Pemanggilan Fungsi

Cara terakhir dalam menjalankan algoritma rekursif adalah berdasarkan eksekusinya. Bagaimana maksudnya? Hal ini berkaitan dengan letak eksekusi pemanggilan fungsi rekursif. Ia dibedakan menjadi dua cara, yaitu tail recursion dan non-tail recursion.

#### Tail Recursion

Fungsi rekursif dapat dikatakan sebagai tail recursion apabila pemanggilan fungsi berada di akhir blok program. Coba perhatikan kode berikut.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35133?from=35128#run6-editor1)

1

def counting\_number(n : int):

2

if n > 0:

3

print(f"{n}", end=" ")

4

counting\_number(n-1) # pemanggilan rekursif berada di akhir blok program

5

​

6

counting\_number(5)

7

print()

8

​

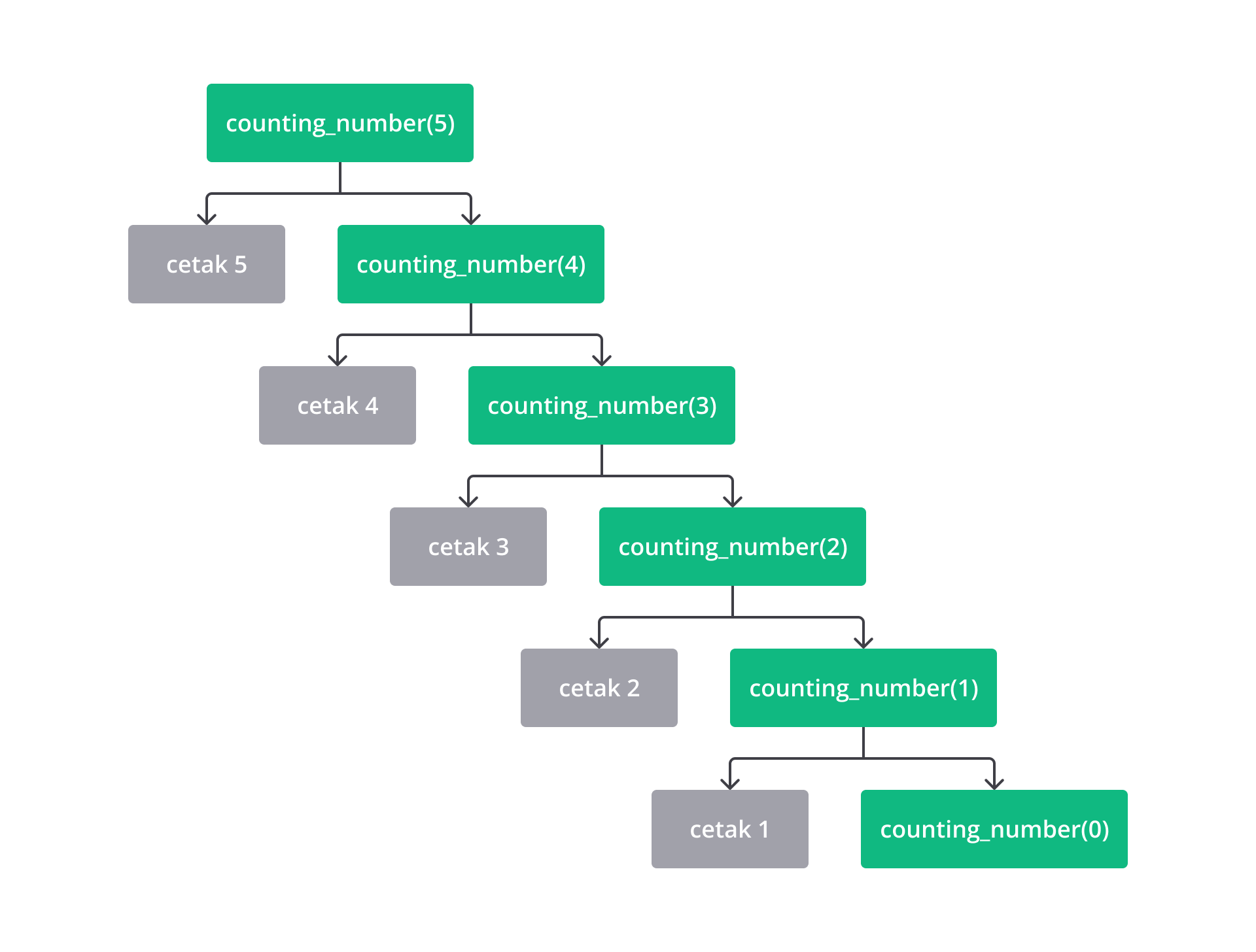
9

# Output: 5 4 3 2 1

 Input  Reset

 Jalankan

Kode di atas adalah contoh algoritma rekursif untuk menampilkan angka yang berurutan mulai dari nilai n hingga 1. Berdasarkan contoh di atas, Anda dapat melihat baris terakhir pada fungsi **counting\_number**. Aksi terakhir yang dijalankan adalah pemanggilan fungsi rekursif. Ia terletak pada akhir blok program. Ia perlu menjalankan aksi lain terlebih dahulu dan diakhiri dengan pemanggilan fungsi rekursif.

[](https://www.dicoding.com/academies/620/tutorials/35133?from=35128)

Berdasarkan pohon pemanggilan fungsi, ia akan mencetak angka 5 dahulu kemudian menjalankan fungsi berikutnya (baca dari kiri ke kanan). Alhasil, output dari program akan mencetak angka 5, 4, hingga 1 secara berurutan.

#### Non-tail Recursion

Kebalikan dari tail recursion, non-tail recursion akan memanggil fungsi rekursif di awal blok program, kemudian dilanjutkan dengan menjalankan aksi lainnya. Coba perhatikan kode berikut.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35133?from=35128#run7-editor1)

1

def counting\_number(n : int):

2

if n > 0:

3

counting\_number(n-1) # pemanggilan rekursif tidak berada di akhir blok program

4

print(f"{n}", end=" ")

5

​

6

counting\_number(5)

7

print()

8

​

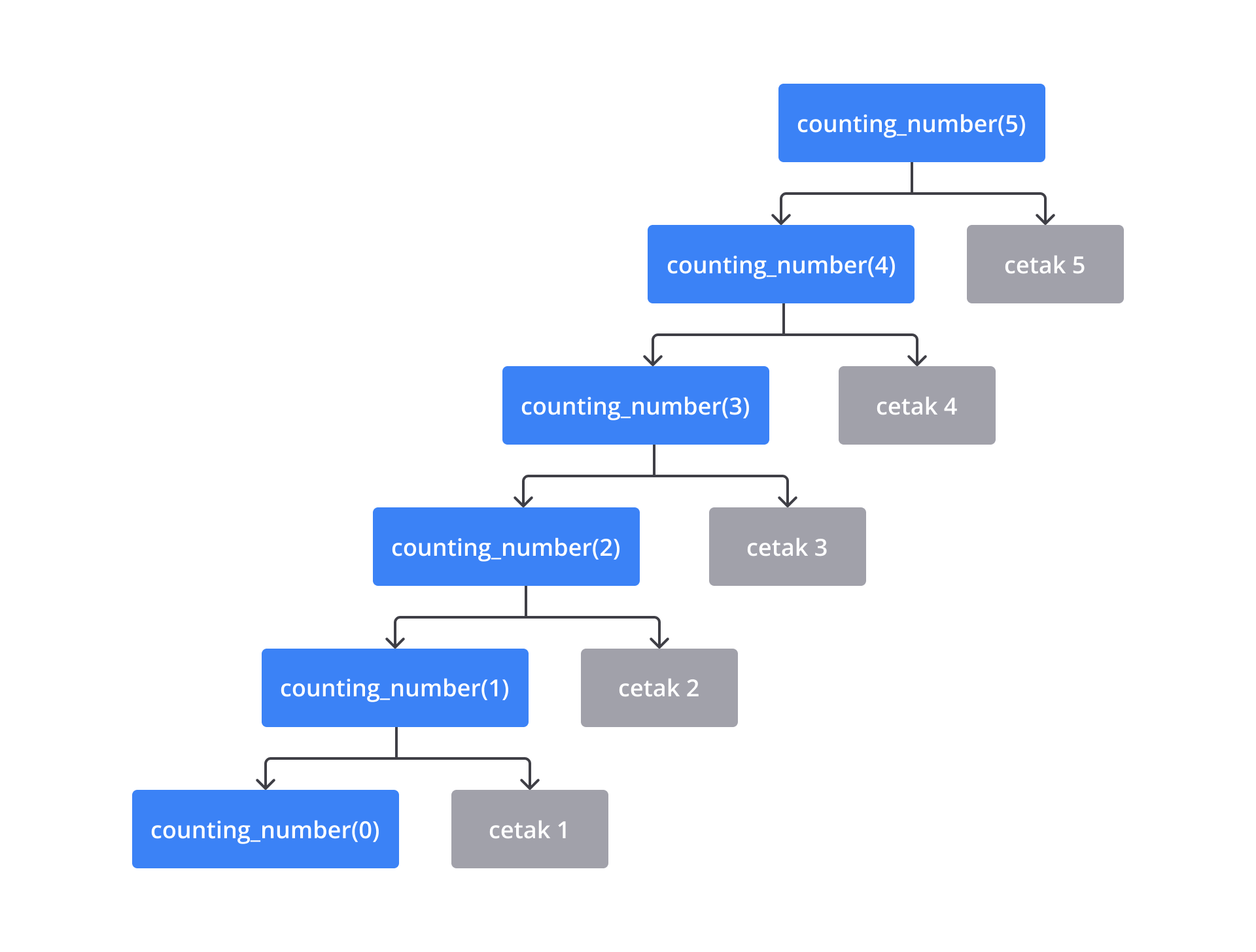
9

# Output: 1 2 3 4 5

 Input  Reset

 Jalankan

Layaknya program sebelumnya, kode di atas menampilkan angka berurutan, tetapi dimulai dari angka 1 hingga n. Program hanya memindahkan pemanggilan fungsi rekursif sebelum aksi mencetak angka. Jika digambarkan dalam ilustrasi, pohon pemanggilan fungsi akan seperti berikut.

[](https://www.dicoding.com/academies/620/tutorials/35133?from=35128)

Dilihat dari susunan pohon pemanggilan fungsi, aksi mencetak angka berada di akhir langkah. Program akan menjalankan fungsi rekursif dahulu hingga kondisi base case terpenuhi. Setelah terpenuhi, aksi mencetak akan dimulai dari bawah hingga ke atas. Alhasil, output program tidak dimulai dari angka 5, tetapi angka 1 dan berakhir pada angka 5.

Materinya menarik sekali, bukan? Saat ini, Anda telah memahami tiga cara untuk menyelesaikan permasalahan dengan algoritma rekursif, yakni berdasarkan struktur, metode, dan eksekusi pemanggilan fungsi. Ketiga cara tersebut memiliki pola yang berbeda sehingga menghasilkan output yang berbeda pula.

Apabila Anda ingin mengeksplorasi materi ini, silakan kunjungi tautan berikut.

* [Data Structures and Algorithms in Python: Recursion](https://learning.oreilly.com/library/view/data-structures-and/9781118290279/09_chap04.html)

## Di Balik Algoritma Rekursif

Sebuah algoritma rekursif memiliki cara yang unik untuk menyelesaikan permasalahan. Ia mampu menyelesaikan dengan cara yang serupa sehingga prosesnya dapat diulang berkali-kali hingga kondisi tertentu. Kita sebagai pengembang aplikasi harus menelaah lebih dalam agar program yang dibuat dapat beroperasi dengan efektif dan efisien.

Nyatanya, ada aspek yang perlu kita pahami di balik algoritma rekursif. Tidak serta merta semua program dialihkan ke konsep rekursif. Adakalanya kita memerlukan konsep rekursif untuk menyederhanakan program, bahkan ada pula yang membuat program semakin rumit.

Untuk itu, mari kita pahami lebih dalam kelebihan dan kekurangan dalam menggunakan konsep rekursif. Simak materinya hingga selesai, ya.

[](https://www.dicoding.com/academies/620/tutorials/35138?from=35133)

### Kelebihan Algoritma Rekursif

Sebagaimana materi yang disampaikan sebelumnya, algoritma rekursif mampu menyederhanakan program yang kompleks. Ia mampu membagi masalah besar menjadi masalah yang kecil-kecil dan diselesaikan satu per satu. Ia memiliki kelebihan yang mampu menjadi daya tarik bagi pengembang aplikasi dalam menyelesaikan masalah kompleks.

Berikut adalah kelebihan yang ditawarkan algoritma rekursif dalam menyelesaikan masalah kompleks.

#### Penyelesaian yang Solutif

Penyelesaian masalah yang menerapkan algoritma rekursif adalah dengan memecah suatu masalah ekstrem menjadi masalah kecil-kecil. Solusi pada masalah kecil dapat diulang sedemikian rupa sehingga menyelesaikan masalah besar. Berdasarkan konsep, hal tersebut bersifat solutif karena dapat menghemat aksi atau baris kode.

Tanpa rekursif, permasalahan dapat diselesaikan dengan mudah. Namun, Anda mampu menyelesaikan permasalahan tersebut dengan rekursif sebagai alternatif lain. Dengan begitu, kita sebagai pengembang aplikasi dapat memperkaya cara berpikir dalam menyelesaikan permasalahan.

Misalnya, kita ingin mencetak angka mulai dari 1 hingga n. Cara apa yang ada di benak kita untuk menyelesaikan masalah tersebut? Mungkin, kita dapat mencetak angka dari 1 hingga nilai n menggunakan perulangan. Dengan konsep rekursif, kita dapat mencetak angka dari nilai 1 hingga n dengan batasan tidak boleh melebihi nol tanpa adanya perulangan. Alhasil, programnya akan seperti berikut.

* [**cetak\_angka\_dengan\_rekursif.py**](https://www.dicoding.com/academies/620/tutorials/35138?from=35133#tab1-code1)
* [cetak\_angka\_tanpa\_rekursif.py](https://www.dicoding.com/academies/620/tutorials/35138?from=35133#tab1-code2)

1. def counting\_number\_1(n : int):
2. if n > 0:
3. counting\_number\_1(n-1)
4. print(f"{n}", end=" ")
6. counting\_number\_1(5)
7. print()
9. # Output: 1 2 3 4 5

#### Penyederhanaan Konsep

Dengan konsep rekursif, permasalahan kompleks dapat dipecah menjadi satu solusi yang sederhana. Ia mampu menyederhanakan permasalahan tersebut agar dapat diulang berkali-kali.

Kita tidak perlu lagi memikirkan konsep iterasi dalam perulangan. Hal itu sudah ditutup dengan konsep rekursif yang mengulang aksi hingga kondisi base case terpenuhi. Kita hanya perlu menggambarkan solusi berdasarkan definisi masalah itu sendiri.

Misalnya, kita ingin menghitung hasil perpangkatan pada suatu angka, contohnya 2 pangkat 5. Tanpa ekspresi matematis, kita dapat menyelesaikan masalah tersebut dengan cara mengalikan 2 sebanyak 5 kali. Hal ini dapat dilakukan menggunakan perulangan dengan menyimpan perkalian dalam suatu variabel. Nyatanya, Anda tidak perlu menyimpan nilai apa pun dalam menyelesaikan masalah itu dengan rekursif.

Jadi, program perpangkatan akan seperti berikut.

* [**perpangkatan\_dengan\_rekursif.py**](https://www.dicoding.com/academies/620/tutorials/35138?from=35133#tab2-code1)
* [perpangkatan\_tanpa\_rekursif.py](https://www.dicoding.com/academies/620/tutorials/35138?from=35133#tab2-code2)

1. def power\_1(base, exponent):
2. if exponent == 0:
3. return 1
4. else:
5. return base \* power\_1(base, exponent - 1)
7. print(power\_1(2, 5))
9. # Output: 32

#### Penggunaan Kode yang Singkat

Penerapan rekursif sering kali memungkinkan kita untuk menulis kode lebih singkat dibandingkan tanpa rekursif. Oleh karena itu, baris kode akan semakin sedikit dan kode lebih mudah dibaca.

Masih ingatkah Anda dengan binary search? Ya, salah satu cara untuk menemukan data dari sekumpulan data. Pada materi sebelumnya, kita mengulik lebih dalam tentang binary search. Kode untuk menyelesaikan permasalahannya cukup panjang, bukan? Dengan konsep rekursif, pencarian data menggunakan binary search akan lebih singkat dan mudah untuk dipahami.

Berikut adalah perbandingan antara kode binary search dengan rekursif dan tanpa rekursif.

* [**binary\_search\_dengan\_rekursif.py**](https://www.dicoding.com/academies/620/tutorials/35138?from=35133#tab3-code1)
* [binary\_search\_tanpa\_rekursif.py](https://www.dicoding.com/academies/620/tutorials/35138?from=35133#tab3-code2)

1. def binary\_search\_1(numbers: list, target: int, index=0) -> int:
2. if index >= len(numbers):
3. return -1
4. elif numbers[index] == target:
5. return index
6. else:
7. return binary\_search\_1(numbers, target, index + 1)
9. # main program
10. numbers = [23,99,55,49,89,72,9,13,42,62]
11. number\_what\_we\_looking\_for = 42
13. numbers = sorted(numbers)
15. result = binary\_search\_1(numbers, number\_what\_we\_looking\_for)
17. if result != -1:
18. print(f"Angka {number\_what\_we\_looking\_for} ditemukan pada indeks ke-{result}.")
19. else:
20. print("Angka tidak ditemukan dalam kumpulan data.")

23. # Output: Angka 42 ditemukan pada indeks ke-3.

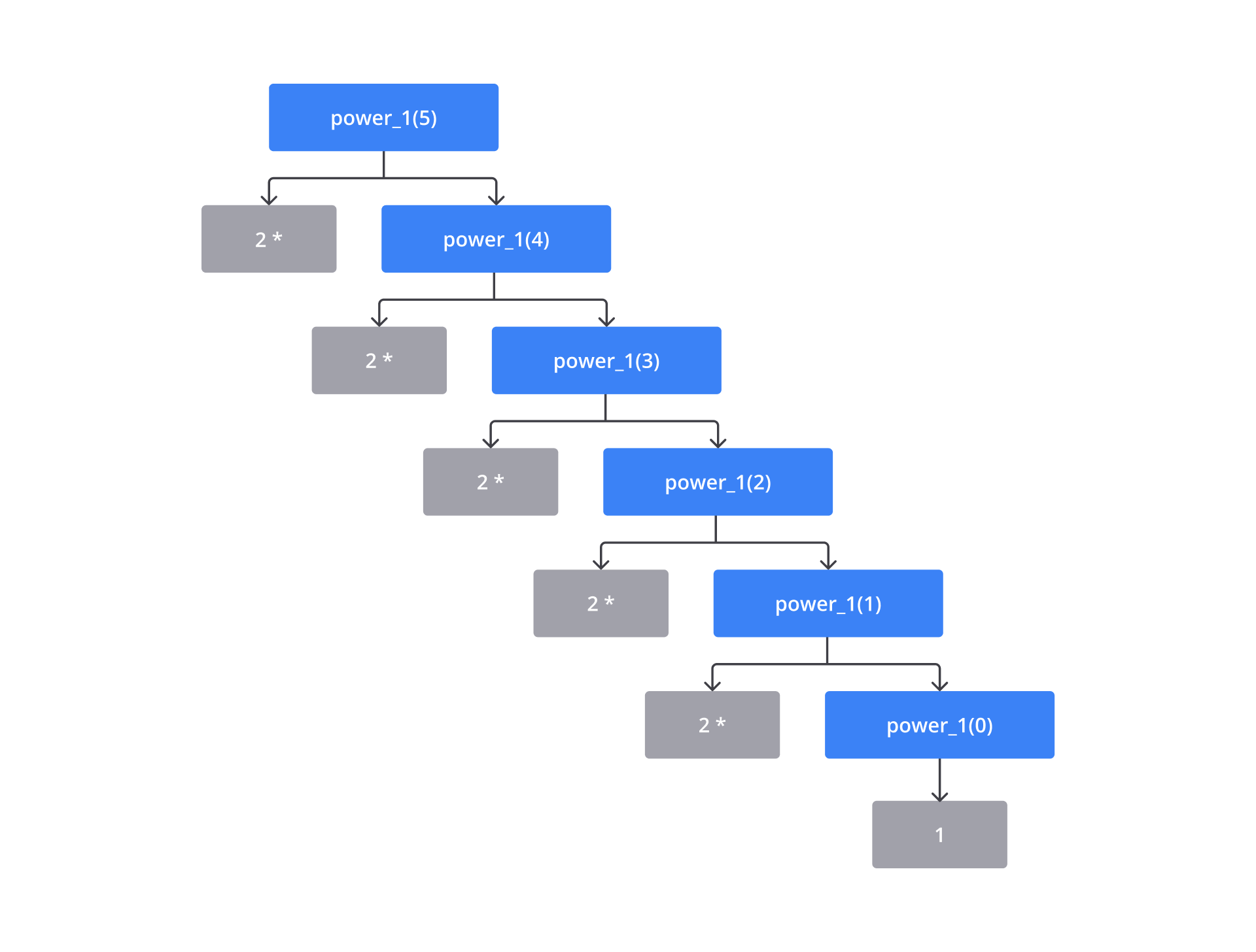
#### Meningkatkan Keterbacaan Kode

Selain penulisan kode yang singkat, keterbacaan kode juga menjadi salah satu kelebihan dalam menerapkan konsep rekursif. Dalam konsep rekursif, pemecahan masalah yang kompleks akan dipecah menjadi sub masalah yang kecil sehingga menciptakan hierarki yang terstruktur. Hal ini memungkinkan pemanggilan dan penulisan kode menjadi lebih singkat dan cenderung lebih mudah dipahami.

Ketika ingin menyelesaikan permasalahan perpangkatan, seperti 2 pangkat 5, kita akan menjabarkannya menjadi dua dikalikan dengan dua sebanyak lima kali atau dapat ditulis menjadi 2 x 2 x 2 x 2 x 2. Penjabaran tersebut dapat diterapkan dalam konsep rekursif sehingga kodenya menjadi seperti berikut.

1. def power\_1(base, exponent):
2. if exponent == 0:
3. return 1
4. else:
5. return base \* power\_1(base, exponent - 1)
7. print(power\_1(2, 5))
9. # Output: 32

Lo, mengapa kode di atas sama seperti penjabaran sebelumnya? Saat fungsi power\_1 pertama kali dipanggil, ia akan mengembalikan nilai dua dikalikan fungsi rekursif dengan nilai eksponen decrement pada setiap pemanggilannya. Hal ini akan terus berulang hingga nilai eksponen bernilai satu. Apabila dijabarkan dan disusun secara pohon pemanggilan fungsi, ia akan terlihat seperti pada ilustrasi berikut.

[](https://www.dicoding.com/academies/620/tutorials/35138?from=35133)

Coba perhatikan ilustrasi di atas. Apabila perkalian disusun secara rapi, ia akan menghasilkan struktur perkalian yang serupa dengan penjabaran sebelumnya, yaitu 2 x 2 x 2 x 2 x 2 x 1. Konsep tersebut akan memudahkan kita untuk membaca, menelaah, dan memahami kode dengan cepat.

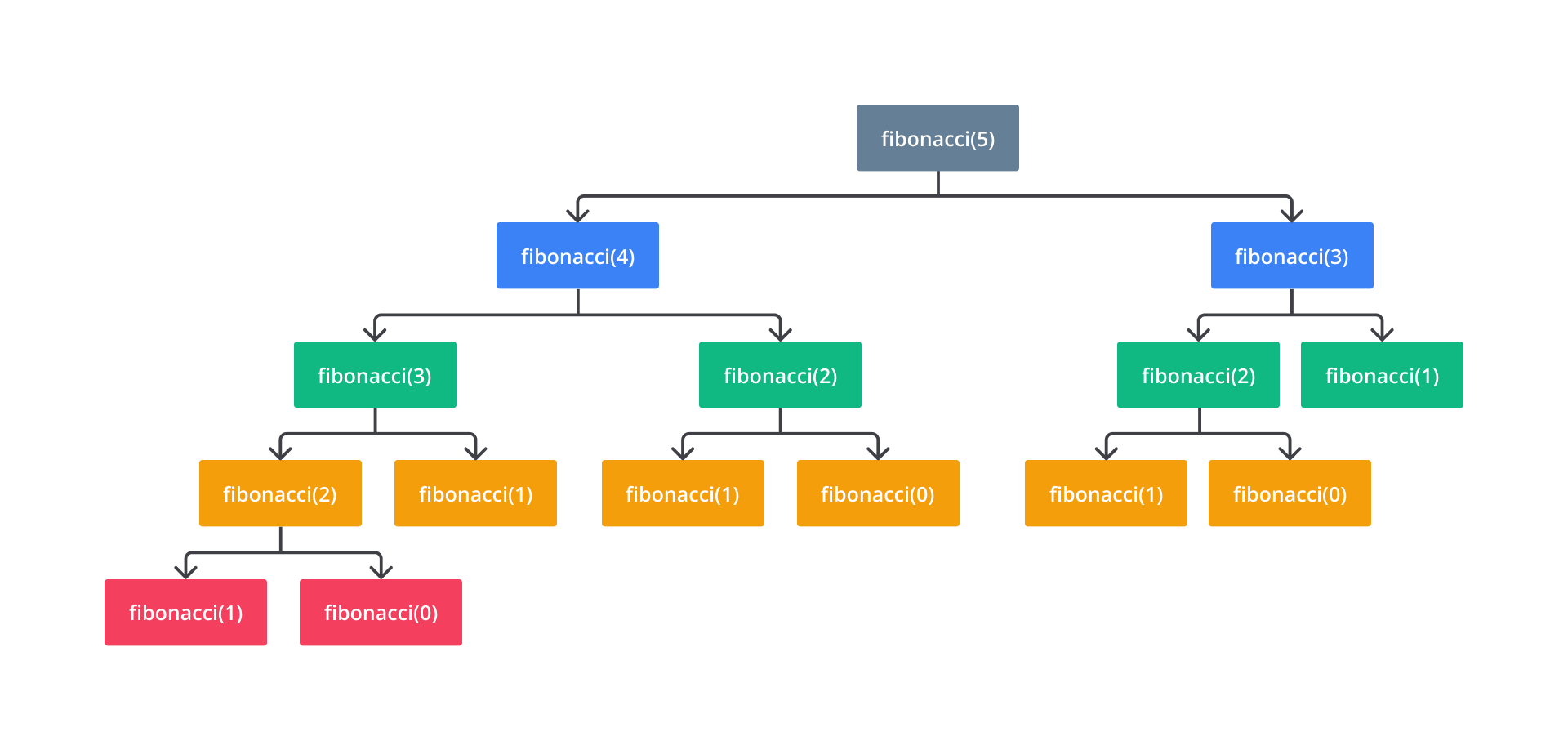
### Kekurangan Algoritma Rekursif

Di balik keunggulan, ternyata algoritma rekursif memiliki kekurangan. Walaupun mampu menyederhanakan masalah yang kompleks, algoritma ini belum efisien menyelesaikan permasalahan. Berikut adalah kekurangan yang ada pada algoritma rekursif dalam menyelesaikan masalah kompleks.

#### Efisiensi yang Rendah

Salah satu kekurangan yang dapat terjadi saat Anda menggunakan konsep rekursif adalah tingkat efisiensi yang rendah. Lo, bagaimana bisa? Terkadang, algoritma rekursif membutuhkan banyak alokasi memori untuk menyimpan panggilan dan nilai. Hal ini dapat membuat program memiliki kinerja yang rendah dan tidak efisien.

Masih ingatkah Anda dengan pohon pemanggilan fungsi pada program fibonacci? Ya, untuk mencari nilai fibonacci pada elemen ke-n, program perlu mengetahui dua elemen yang saling berdekatan dengan indeks tersebut. Tidak hanya itu, untuk menemukan dua elemen tersebut dibutuhkan komputasi yang berulang-ulang pada setiap sisi. Hal ini yang membuat program mengulangi suatu aksi berkali-kali.

[](https://www.dicoding.com/academies/620/tutorials/35138?from=35133)

Dari bagan di atas, berapa kali fibonacci(2) dihitung? Untuk mencari elemen fibonacci ke-5, program perlu menghitung fibonacci(2) sebanyak tiga kali. Bagaimana dengan elemen ke-100? Program akan mengulangi proses fibonacci(2) berkali-kali. Sebenarnya, perhitungan tersebut untuk mencari elemen berikutnya saja, tetapi selalu dipanggil berkali-kali. Hal inilah yang membuat algoritma rekursif tersebut kurang efisien sehingga perlu diselesaikan dengan pendekatan lain, seperti dynamic programming (nanti akan dibahas pada akhir materi).

#### Penggunaan Waktu yang Tidak Efektif

Melihat aspek sebelumnya, beberapa algoritma rekursif dapat memiliki kinerja yang buruk. Kondisi tersebut disebabkan oleh pemanggilan fungsi dan penyimpanan data yang dilakukan berkali-kali. Semakin banyak fungsi yang dipanggil, waktu eksekusi program akan semakin tinggi. Itulah alasan algoritma rekursif memiliki efisiensi komputasi yang rendah.

Mengingat program rekursif fibonacci yang sering memanggil fungsi secara berulang, waktu eksekusi akan semakin meningkat. Mungkin, untuk menemukan elemen ke-5 dari deret fibonacci terlihat cepat. Akan tetapi, bagaimana dengan elemen ke-100, bahkan ke-1000? Aksi yang dijalankan semakin banyak menyebabkan waktu komputasinya lebih lama juga.

#### Proses Debugging yang Sulit

Pada iterasi perulangan, kita mampu mengidentifikasi state yang sedang berulang. Hal ini terjadi karena hanya satu blok program yang dijalankan pada setiap iterasinya. Apabila kesalahan terjadi dalam aksi tertentu pada iterasi kesekian, kita dapat menganalisisnya.

Berbeda dengan rekursif yang memungkinkan program menjalankan blok program lebih dari satu. Kondisi ini dapat terjadi saat tree recursion berjalan. Kita akan sulit melacak dan memperbaiki bug karena melibatkan pemanggilan fungsi diri sendiri yang terus berulang. Alhasil, jejak panggilan sulit untuk dilacak.

Selain dari sisi blok program, algoritma rekursif biasanya tidak menyimpan state atau variabel. Ia secara langsung memanggil fungsi rekursif tanpa menyimpan nilai apa pun. Hal ini membuat proses rekursif sulit untuk dianalisis karena tidak ada jejak nilai yang tertinggal.

Perhatikan program fibonacci berikut.

1. def fibonacci(n: int) -> int:
2. if n == 0:
3. return 0
4. elif n == 1:
5. return 1
6. else:
7. return fibonacci(n-1) + fibonacci(n-2)
9. n = 5
10. hitung\_fibonacci = fibonacci(n)
11. print(f"Fibonacci dari {n} adalah {hitung\_fibonacci}.")
13. # Output: Fibonacci dari 5 adalah 5.

Apakah Anda melihat proses assignment pada fungsi fibonacci? Tidak ada, bukan? Belum lagi saat kita salah memberikan kondisi base case. Infinite loop menjadi hal yang tidak bisa kita hindari. Hal ini yang menyebabkan output program sulit untuk dianalisis.

#### Tidak Cocok untuk Semua Masalah

Tidak semua masalah dapat diselesaikan dengan pendekatan rekursif walaupun ia memiliki berbagai keunggulan. Beberapa masalah mungkin lebih cocok untuk solusi iteratif atau pendekatan lain.

Coba perhatikan program fibonacci. Pada saat membaca ini, Anda akan merasa bahwa program fibonacci tidaklah cocok untuk diselesaikan dengan konsep rekursif. Ia memiliki kinerja yang lambat, waktu komputasi yang tinggi, dan proses debugging yang menyulitkan pengembang aplikasi.

Ternyata, program fibonacci dapat diselesaikan dengan pendekatan dynamic programming. Apa itu? Dynamic programming adalah suatu pendekatan untuk menyelesaikan masalah kompleks dengan memecahnya menjadi sub-masalah kecil layaknya rekursif. Ia memanfaatkan iteratif menggunakan looping untuk menyimpan hasil dari sub-masalah tersebut.

Lalu, apa bedanya dengan pendekatan rekursif? Bedanya, ia dapat menghindari perhitungan berulang. Dengan begitu, ia tidak akan mengalkulasi proses yang sama dan program semakin efisien. Berikut adalah perbandingan antara program fibonacci menggunakan dynamic programming dan rekursif.

* [**fibonacci\_dengan\_perulangan.py**](https://www.dicoding.com/academies/620/tutorials/35138?from=35133#tab4-code1)
* [fibonacci\_dengan\_rekursif.py](https://www.dicoding.com/academies/620/tutorials/35138?from=35133#tab4-code2)

1. def fibonacci\_loop(n: int) -> int:
2. numbers: list = []
3. for i in range(0, n+1):
4. if i <= 1:
5. numbers.append(i)
6. else:
7. next\_fib = numbers[i-1] + numbers[i-2]
8. numbers.append(next\_fib)
9. return numbers[-1]
11. n = 5
12. hitung\_fibonacci = fibonacci\_loop(n)
13. print(f"Fibonacci dari {n} adalah {hitung\_fibonacci}.")
15. # Output: Fibonacci dari 5 adalah 5.

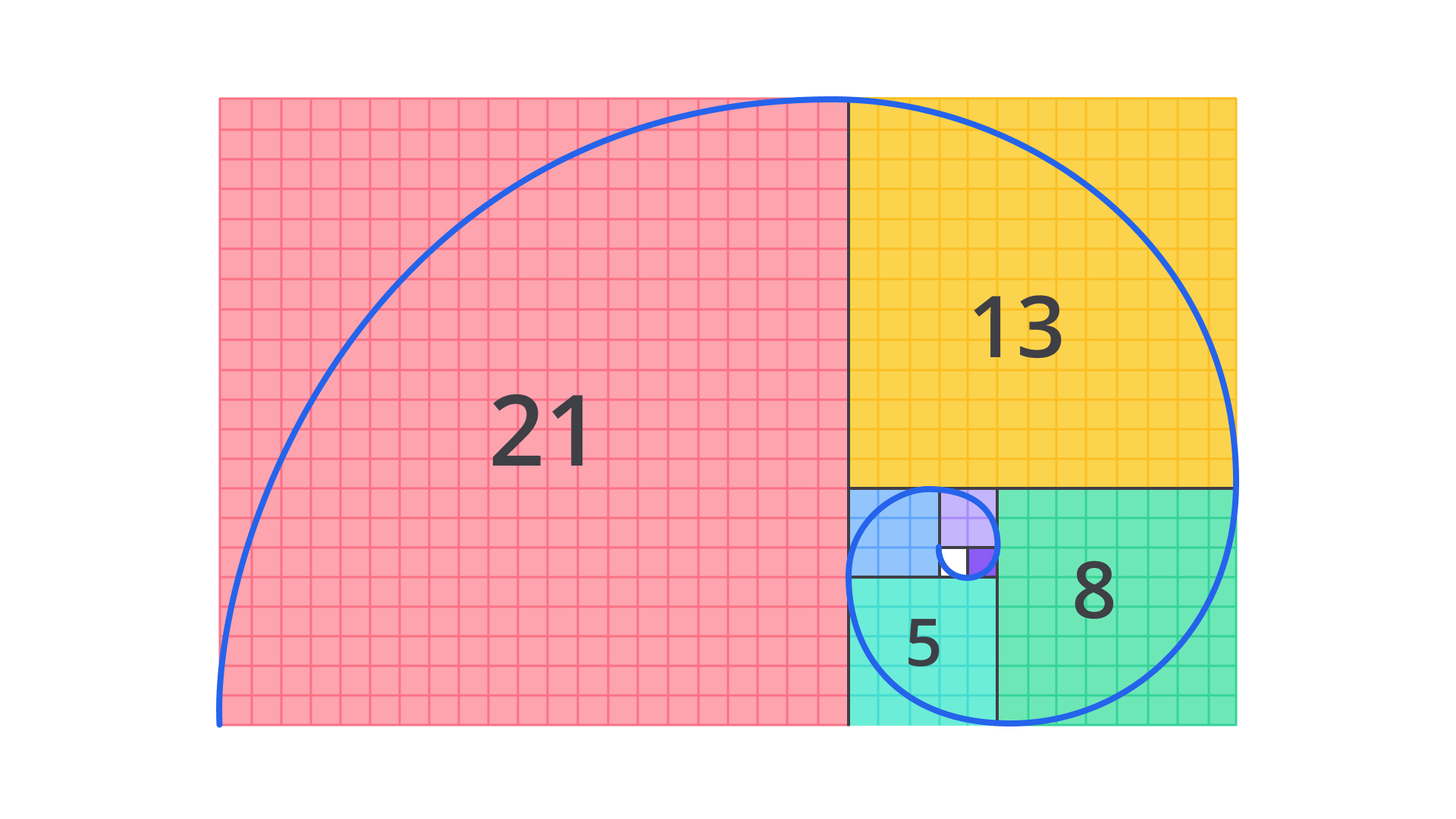
Secara logika, kedua program tersebut terlihat serupa. Akan tetapi, proses mencari dua elemen sebelumnya yang berbeda. Mengapa demikian? Dengan perulangan, kita dapat menyimpan proses perhitungan dalam sebuah variabel numbers. Kita mampu memanggil dua elemen sebelumnya dengan mudah supaya menghasilkan elemen yang diinginkan.

Berbeda dengan program rekursif. Kita perlu mencari dua elemen sebelum menghitungnya mulai dari awal. Ia tidak menyimpan nilai pada variabel apa pun. Alhasil, komputasinya semakin banyak, kinerjanya semakin lambat, dan prosesnya semakin lama.

Sampai saat ini, Anda telah belajar banyak hal terkait algoritma rekursif. Kelebihan dan kekurangan pada algoritma rekursif sudah Anda pelajari walaupun tidak semua program rekursif memiliki karakteristik tersebut. Pemahaman tentang hal ini akan semakin melatih kita untuk memahami waktu yang tepat dalam menggunakan algoritma rekursif. Dengan demikian, kita semakin mantap dalam mengembangkan solusi yang efektif dan efisien.

## Rangkuman Menaklukkan Algoritma Rekursif

Selamat! Anda sudah berada di penghujung materi terkait Menaklukkan Algoritma Rekursif. Sejauh ini, Anda telah belajar arti dari algoritma rekursif, komponen penyusunnya, jenisnya, hingga kelebihan dan kekurangannya. Berikut adalah ringkasan materi yang sudah Anda pelajari.

* Dalam konteks pemrograman, algoritma rekursif adalah suatu fungsi atau prosedur yang berisi aksi untuk memanggil fungsi itu sendiri. Hal tersebut akan berulang terus-menerus hingga kondisi tertentu terpenuhi.
* Algoritma rekursif dapat diartikan bahwa setiap aksi diulang terus-menerus dengan masukan yang berasal dari output proses sebelumnya. Aksi tersebut akan berhenti ketika mencapai kondisi tertentu.
* Ada empat komponen penyusun algoritma rekursif. Berikut komponennya.
  + **Proses pemecahan masalah**: masalah utama dapat diselesaikan dengan menyederhanakannya menjadi masalah yang lebih kecil dan serupa.
  + **Pembuatan fungsi**: algoritma rekursif membutuhkan fungsi agar ia dapat memanggil dirinya sendiri.
  + **Pemanggilan fungsi rekursif**: aksi memanggil dirinya sendiri untuk memecahkan masalah yang kecil dari masalah utama.
  + **Base case**: kondisi atau batasan yang harus didefinisikan supaya proses rekursi berhenti atau tidak memanggil fungsi lagi.
* Deret fibonacci adalah salah satu penerapan algoritma rekursif. Ia dapat dibangun secara sekuensial dengan membuat sebuah fungsi yang memanggil fungsi itu sendiri.
* Berikut adalah kode untuk menghitung elemen terakhir dari fibonacci.
  + def fibonacci(n: int) -> int:
  + if n == 0:
  + return 0
  + elif n == 1:
  + return 1
  + else:
  + return fibonacci(n-1) + fibonacci(n-2)
  + n = 5
  + hitung\_fibonacci = fibonacci(n)
  + print(f"Fibonacci dari {n} adalah {hitung\_fibonacci}.")
  + # Output: Fibonacci dari 5 adalah 5.
* Deret fibonacci memiliki keunikan tersendiri. Ia memiliki proporsi unik antara dua angka yang berurutan. Dalam dunia matematis, proporsi ini disebut sebagai **golden ratio** (rasio emas).  
  [](https://www.dicoding.com/academies/620/tutorials/35148)
* Deret fibonacci atau golden ratio dapat Anda temukan pada berbagai aspek dalam kehidupan nyata. Mulai dari jumlah pertumbuhan batang pohon, daun tumbuhan, benih bunga matahari, bunglon, desain grafis, ombak, dan masih banyak lainnya yang mengandung unsur fibonacci atau golden ratio.
* Jenis algoritma rekursif dibedakan menjadi tiga bagian. Setiap bagian dibagi menjadi dua macam. Berikut pembagiannya.
  + Rekursif berdasarkan **struktur pemanggilan fungsi**.
    - **Linear recursion**: suatu fungsi rekursif yang hanya memanggil satu kali fungsi dalam setiap pemanggilannya.
    - **Tree recursion**: suatu fungsi rekursif yang memanggil fungsi itu sendiri lebih dari satu kali.
  + Rekursif berdasarkan **metode pemanggilan fungsi**.
    - **Direct recursion**: konsep rekursif yang memanggil fungsi secara langsung.
    - **Indirect recursion**: konsep rekursif yang menjalankan fungsi rekursif secara tidak langsung atau membutuhkan fungsi tambahan supaya proses rekursif berjalan.
  + Rekursif berdasarkan **eksekusi pemanggilan fungsi**.
    - **Tail recursion**: fungsi rekursif yang memanggil fungsi di akhir blok program.
    - **Non-tail recursion**: fungsi rekursif yang memanggil fungsi di awal atau tengah blok program.
* Berikut adalah **kelebihan**yang ditawarkan algoritma rekursif dalam menyelesaikan masalah kompleks.
  + **Penyelesaian yang solutif**: dalam beberapa kasus, algoritma rekursif dapat menghasilkan solusi yang sederhana.
  + **Penyederhanaan konsep**: penerapan algoritma rekursif mampu membantu pengembang aplikasi dalam memahami konsep rekursif secara komprehensif.
  + **Penggunaan kode yang ringkas**: implementasi rekursif memungkinkan pengembang aplikasi menuliskan kode lebih singkat daripada solusi iteratif yang kompleks.
  + **Meningkatkan keterbacaan kode**: penulisan kode menjadi lebih singkat dan cenderung lebih mudah dipahami.
* Berikut adalah **kekurangan**yang perlu Anda pahami saat menerapkan algoritma rekursif pada penyelesaian masalah.
  + **Efisiensi yang rendah**: beberapa algoritma rekursif memiliki komputasi yang kompleks sehingga menyebabkan performanya buruk.
  + **Penggunaan waktu yang tidak efisien**: beberapa algoritma rekursif mempunyai komputasi banyak yang menyebabkan efisiensinya rendah.
  + **Proses debugging yang sulit**: proses debugging pada beberapa algoritma rekursif lebih rumit dibandingkan dengan solusi iteratif.
  + **Tidak cocok untuk semua masalah**: tidak serta-merta semua masalah dialihkan ke konsep rekursif. Beberapa masalah mungkin lebih cocok dengan solusi iteratif atau pendekatan lain daripada rekursif.