**Definisi Subprogram**

Setiap program yang Anda bangun, pada akhirnya akan semakin besar seiring dengan kompleksitas masalah yang perlu diselesaikan. Semakin besar sebuah program, bagian kode yang berulang akan bertambah sehingga tidak efisien jika Anda perlu mengetik ulang atau bahkan melakukan ***copy-paste*.**Salah satu kode yang sering berulang adalah rumus atau formula, perhatikan kode di bawah ini.

|  |
| --- |
| # Luas pertama  panjang = 5  lebar = 10  luas\_persegi\_panjang = panjang \* lebar  print(luas\_persegi\_panjang)  # Luas kedua  panjang = 4  lebar = 15  luas\_persegi\_panjang = panjang \* lebar  print(luas\_persegi\_panjang)  Output:  50  60 |

Kode di atas merupakan program untuk mencari luas persegi panjang. Perhatikan bahwa kita perlu menuliskan dua rumus kode yang sama untuk mencari luas persegi panjang dengan nilai panjang dan lebar berbeda.

Lalu, apakah ada cara untuk menghindari kode yang perlu diketik berulang, dan sebaliknya, dapat digunakan berkali-kali? Jawabannya adalah ada, inilah yang disebut sebagai subprogram dan salah satu jenisnya adalah fungsi. Bandingkan kode sebelumnya dengan kode berikut.

|  |
| --- |
| def mencari\_luas\_persegi\_panjang(panjang,lebar):  luas\_persegi\_panjang = panjang\*lebar  return luas\_persegi\_panjang  persegi\_panjang\_pertama = mencari\_luas\_persegi\_panjang(5,10)  print(persegi\_panjang\_pertama)  persegi\_panjang\_kedua = mencari\_luas\_persegi\_panjang(4,15)  print(persegi\_panjang\_kedua)  Output:  50  60 |

Kode di atas merupakan program yang sama dengan sebelumnya dan bertujuan untuk mencari luas persegi panjang. Namun, kali ini kita menggunakan sebuah konsep yang disebut fungsi. Blok fungsi pada kode di atas dimulai dari "def" hingga "return".

Ini adalah tujuan akhir dari materi kali ini, Anda diharapkan memahami subprogram yang di antaranya adalah fungsi dan prosedur.

Subprogram adalah serangkaian instruksi dirancang untuk melakukan operasi yang sering digunakan dalam suatu program. Subprogram yang sering digunakan terdiri dari dua jenis, yakni berikut.

1. **Fungsi**  
   Fungsi adalah blok kode yang dapat menerima input, melakukan pemrosesan, dan mengembalikan output. Hasil atau output tersebut dinyatakan dalam sebuah tipe data yang eksplisit. Artinya, fungsi yang dibuat dapat ditentukan untuk mengembalikan tipe data integer, string, atau lainnya.
2. **Prosedur**  
   Prosedur adalah deretan instruksi yang jelas keadaan awal (*initial state*) dan keadaan akhirnya (*final state*). Prosedur mirip dengan program secara umum, tetapi memiliki cakupan yang kecil dan terbatas.

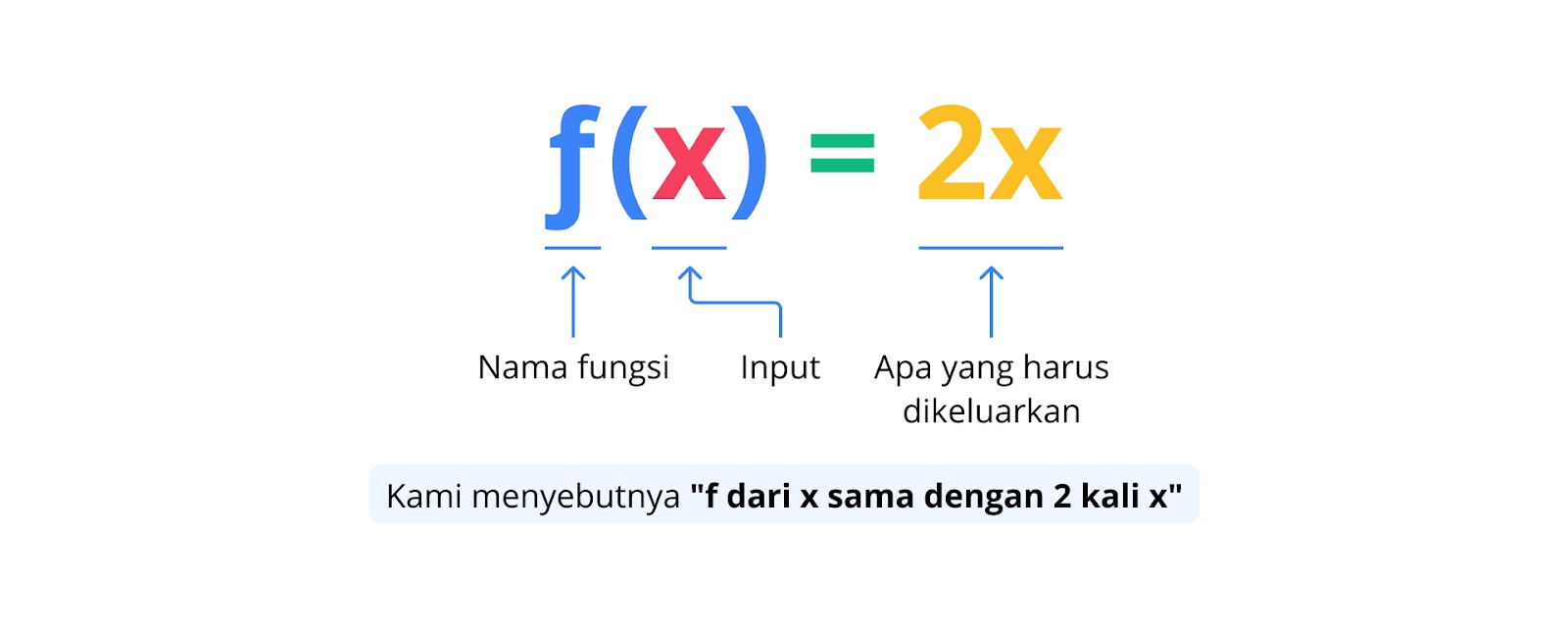
**Fungsi**

Sebelum mulai mempelajari fungsi dalam Python, mari kita bahas terlebih dahulu tentang konsep dasar fungsi.

**Fungsi dalam Matematika**

Fungsi dalam pemrograman sebenarnya didasari oleh konsep pemetaan (asosiasi) dan fungsi dalam matematika. Fungsi pada matematika merupakan pemetaan antara dua himpunan nilai, yaitu **domain**dan **range**. Kita bisa bayangkan fungsi sebagai sebuah mesin yang memiliki input(**domain**)dan *output*(**range**). **Output**tersebut pasti terkait dengan **input** bagaimana pun kondisinya.

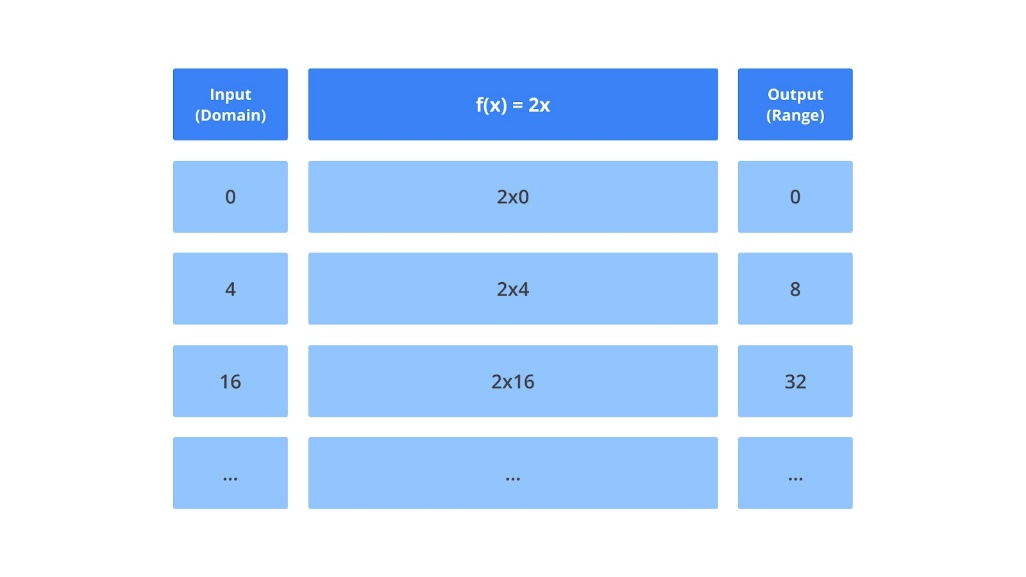
Notasi atau bentuk rumus fungsi dalam matematika beragam. Salah satu yang umum dijumpai adalah notasi berikut.



Dengan catatan sebagai berikut.

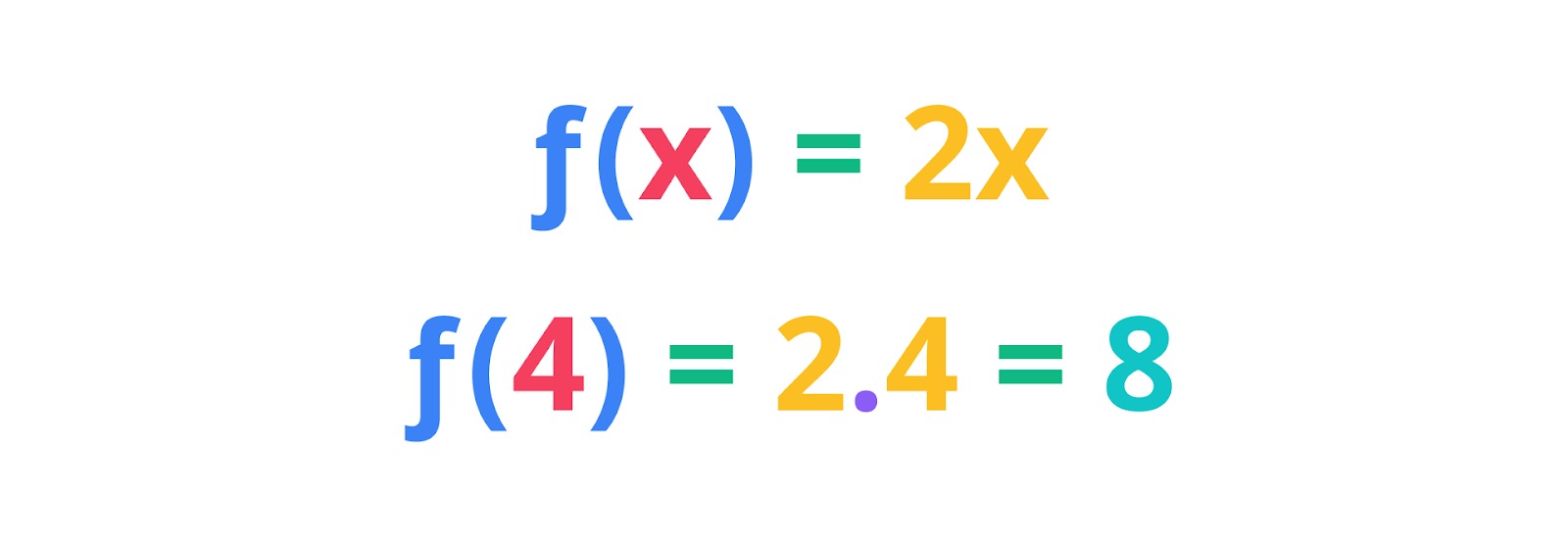
1. f = nama fungsi
2. x = input
3. 2x = apa yang harus dikeluarkan (output)

Mari kita lihat salah satu soal fungsi sederhana dalam matematika. Asumsikan bahwa kita memiliki sebuah fungsi untuk **mengalikan bilangan bulat dengan nilai 2**. Jadi, berikut perhitungannya.



Perhatikan pada gambar tersebut. Sebab fungsi kita bertujuan untuk **mengalikan bilangan bulat dengan nilai 2**, setiap elemen yang berada pada **himpunan domain**akan dikalikan dua dan hasilnya ditampung pada **himpunan range**.

Notasi **f(x)=2x** menunjukkan bahwa fungsi "f" mengambil "x" dan mengalikannya dengan 2.



**Fungsi dalam Pemrograman**

Fungsi dalam pemrograman didasari oleh fungsi dalam matematika. Jadi, baik fungsi pemrograman maupun fungsi matematika memiliki tujuan yang sama, yaitu **mengubah suatu bentuk menjadi bentuk lain dengan aturan yang sama.**

Mari analogikan kembali dengan konsep lain, kita bisa umpamakan fungsi dalam pemrograman seperti merakit isi **black box**.



Selayaknya **black box**, kita tidak perlu tahu tentang hal yang terjadi dalam kotak (fungsi) tersebut. Kita hanya perlu fokus pada keadaan awal yang merupakan himpunan nilai yang terdefinisi sebagai input (domain) dan keadaan akhir yang merupakan himpunan nilai yang terdefinisi sebagai output (range).

Mari ambil contoh salah satu fungsi yang sedari materi awal sudah kita kenal, yakni fungsi "print()". Fungsi ini bertujuan untuk menampilkan teks ke layar tanpa kita perlu ketahui hal yang terjadi di dalamnya. Hal yang perlu kita lakukan adalah memasukkan teks yang diinginkan ke fungsi "print()".

Jadi, kalau kita definisikan, fungsi dalam pemrograman adalah blok kode yang dapat digunakan kembali untuk mengeksekusi fungsionalitas tertentu saat dipanggil. Dalam Python, fungsi terbagi menjadi dua jenis.

1. ***Built-in Functions***  
   *Built-in functions* atau dalam bahasa Indonesia berarti fungsi bawaan adalah kumpulan fungsi yang sudah terintegrasi dengan bahasa pemrograman Python sehingga tidak perlu mengimpor **modul** atau **library** tambahan. Fungsi bawaan ini menyediakan fungsi-fungsi inti dan dasar dari bahasa Python. Contoh dari fungsi bawaan adalah **print(), len(), type(), range(),**dan sebagainya.
2. ***User-defined Functions***  
   *User-defined functions* atau dalam bahasa Indonesia berarti fungsi yang didefinisikan pengguna adalah jenis fungsi yang kita definisikan sendiri untuk melakukan tugas spesifik tertentu. Contoh dari **user-defined functions**adalah fungsi yang telah kita buat di awal materi ini tentang mencari luas persegi panjang.

Jika ingin menggunakan fungsi lain di luar dari **built-in functions**, Anda bisa mengimpor sebuah library. **Library**adalah koleksi dari banyaknya modul yang saling terkait dan dapat digunakan berulang kali. **Modul**adalah file berisi kode Python berupa fungsi, kelas, dan sebagainya.

Library dalam Python terdiri dari dua jenis.

1. **Python Standard Library**  
   Python Standard Library adalah jenis library yang telah terpasang secara otomatis ketika Anda melakukan instalasi Python. Python Standard Library berisi kumpulan modul dan paket yang disertakan secara default oleh Python. Paket (package) merupakan sebuah direktori berisi satu atau lebih modul yang terkait dan saling berhubungan.  
     
   Anda tidak perlu melakukan instalasi untuk menggunakan Python Standard Library. Contoh Python Standard Library adalah **"os"**, **"datetime"**, **"re"**, dan sebagainya. Anda bisa melihat banyaknya jenis library ini pada laman website [berikut](https://docs.python.org/3/library/" \t "_blank).
2. **External Library**  
   Jika sebelumnya impor library tidak perlu dilakukan untuk Python Standard, berbeda halnya dengan external library yang mengharuskan Anda mengimpor library untuk bisa menggunakannya. External Library adalah jenis library yang dikembangkan oleh individu atau organisasi di luar tim inti pengembang Python.  
     
   Sederhananya, di luar sana banyak developer yang turut membuat kode untuk diri mereka sendiri dan pada akhirnya disebarluaskan untuk digunakan oleh developer lainnya. Contoh dari external library adalah TensorFlow yang merupakan library populer untuk menyelesaikan permasalahan pemelajaran mesin (*machine learning*).

**Fungsi**

Blok kode yang dapat digunakan kembali untuk mengeksekusi fungsionalitas tertentu saat dipanggil.

print(), len(), mencari\_luas\_persegi\_panjang()

**Built-in functions**

Kumpulan fungsi yang sudah terintegrasi dengan bahasa pemrograman Python sehingga tidak perlu mengimpor modul atau library tambahan.

print(), len(), range()

**User-defined functions**

Jenis fungsi yang kita definisikan sendiri untuk melakukan tugas spesifik tertentu.

mencari\_luas\_persegi\_panjang()

**Modul**

File berisi kode Python berupa fungsi, kelas, dan sebagainya.

Math, dan semua file yang kita buat sendiri dengan ekstensi ".py" (main.py, var.py, dan lain sebagainya)

**Package**

Sebuah direktori berisi satu atau lebih modul yang terkait dan saling berhubungan.

NumPy, Pandas

**Library**

Koleksi dari banyaknya modul dan paket yang saling terkait dan dapat digunakan berulang kali.

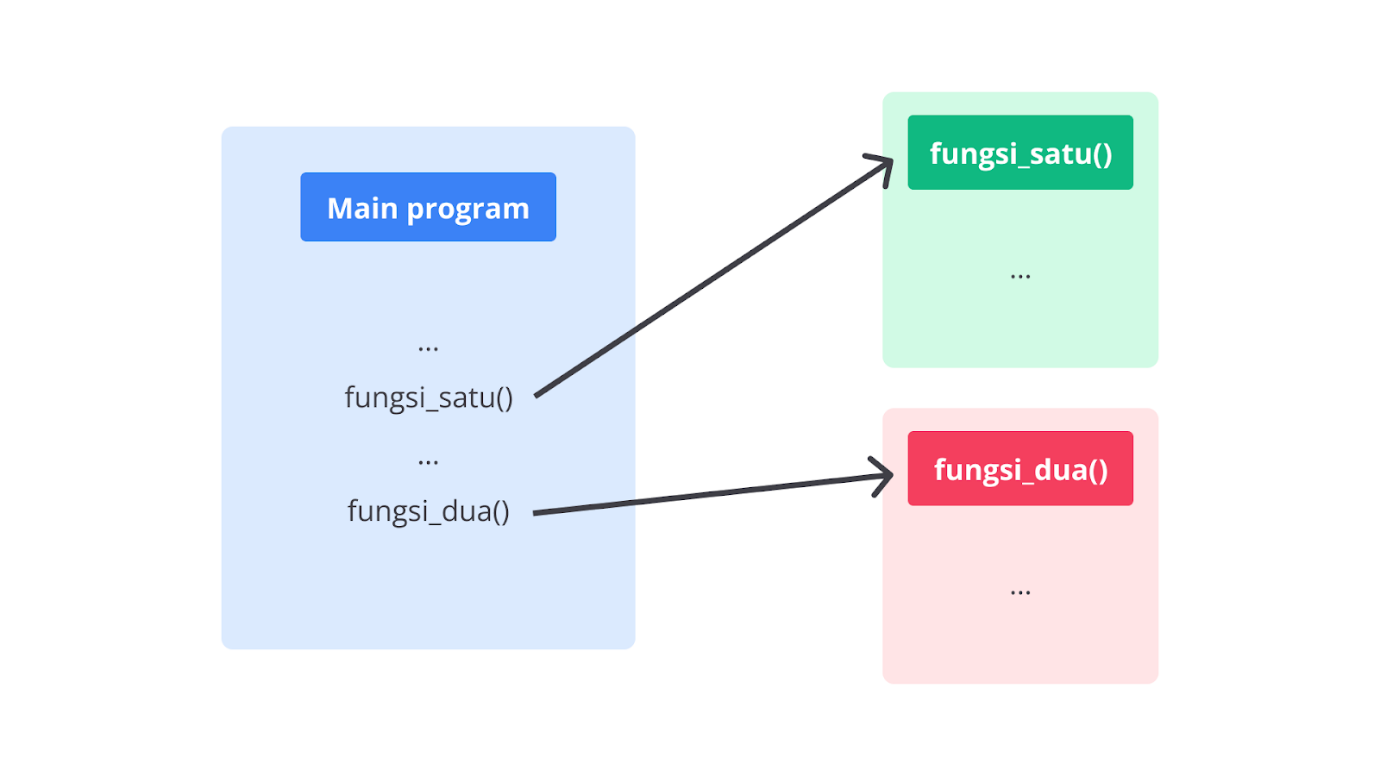
Matplotlib, TensorFlow, Beautiful Soup

Kerap kali kita salah mengartikan library, package, dan modul. Ketiga hal tersebut sebenarnya saling berkaitan. Misalnya, terkadang ketika menyebut NumPy sebagai library, sebenarnya itu sah-sah saja karena library berisi package dan juga modul.

**Kegunaan Fungsi**

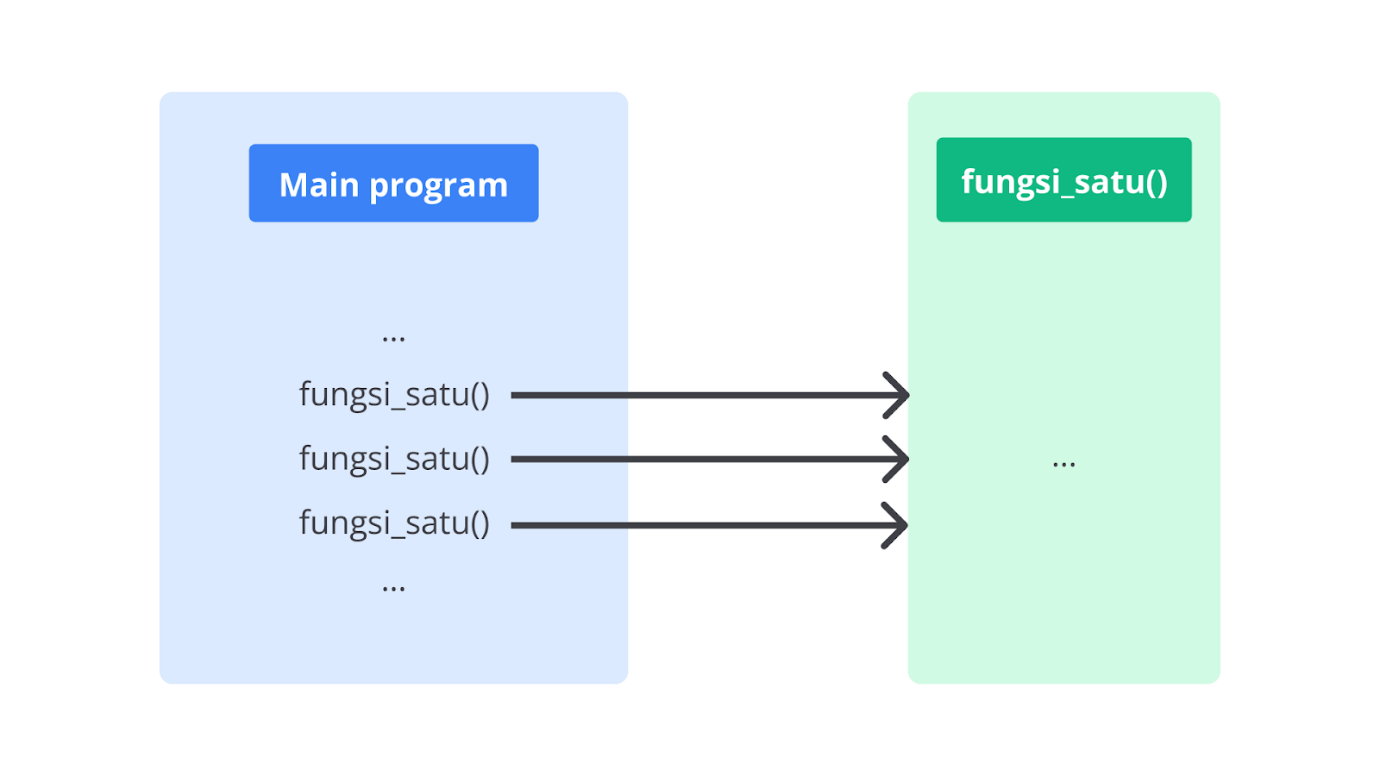
Mengapa kita harus menggunakan fungsi dalam pemrograman? Sebenarnya terdapat banyak sekali kegunaan fungsi yang dapat menyelesaikan masalah pada pemrograman. Beberapa kegunaannya sebagai berikut.

1. Program dapat dipecah menjadi bagian yang lebih kecil (sub).



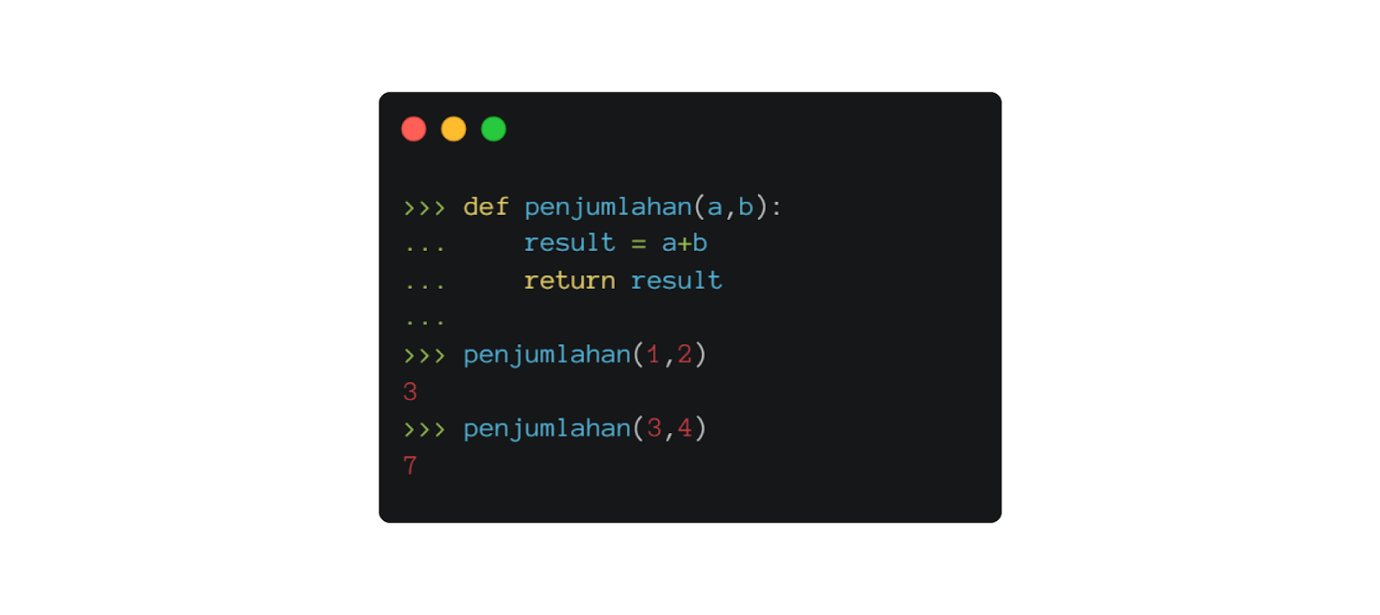
Ketika membuat kode program yang kompleks, Anda bisa membagi setiap programnya menjadi bagian-bagian kecil dengan mendefinisikan fungsi, dan di dalamnya setiap fungsi dapat dipanggil sebagai satu baris atau ekspresi dalam program utama.

1. Penggunaan ulang kode alih-alih menulis ulang kode.



Ketika Anda merasa perlu membuat kode yang berulang-ulang, pemrograman akan menjadi lebih efisien jika kode tersebut diorganisasikan sebagai sebuah fungsi. Contohnya fungsi untuk mencari luas persegi panjang akan sangat berguna dalam berbagai jenis persoalan dengan nilai panjang dan lebar yang berbeda.

1. Setiap fungsi bersifat independen dan dapat diuji secara terpisah.

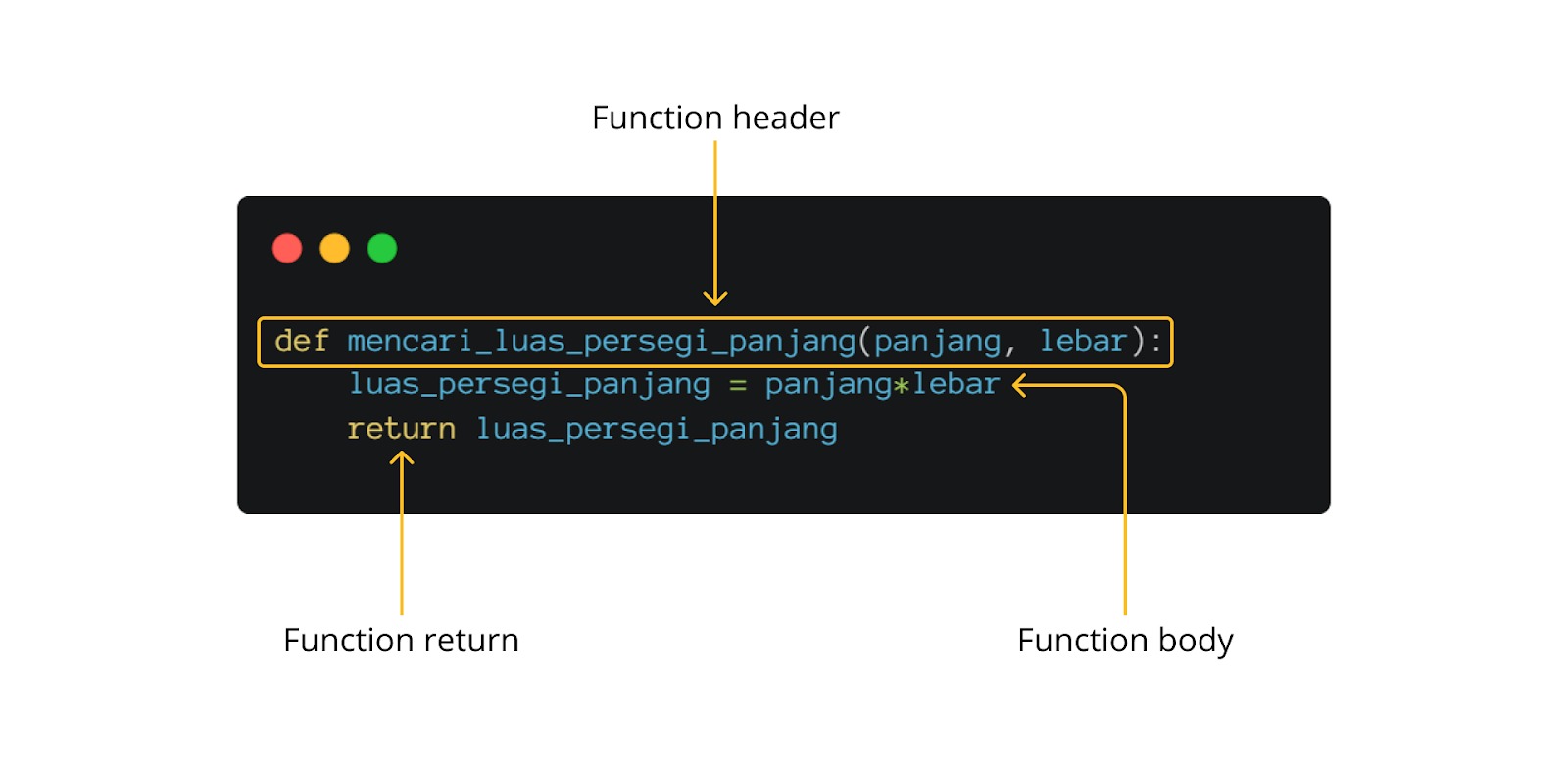


Setiap fungsi bersifat independen, artinya Anda dapat menguji setiap fungsi tersebut dalam interpreter (mode interaktif Python) tanpa perlu membuat program utuh terlebih dahulu. Ketika bekerja dalam program yang lebih kompleks dan melibatkan banyak programmer, hal ini juga mempermudah pembagian tugas masing-masing.

**Mendefinisikan Fungsi dalam Python**

Baiklah, Anda sudah memahami secara mendasar tentang fungsi dan kegunaannya. Mari kita mulai membuat fungsi sendiri (*user-defined functions*).

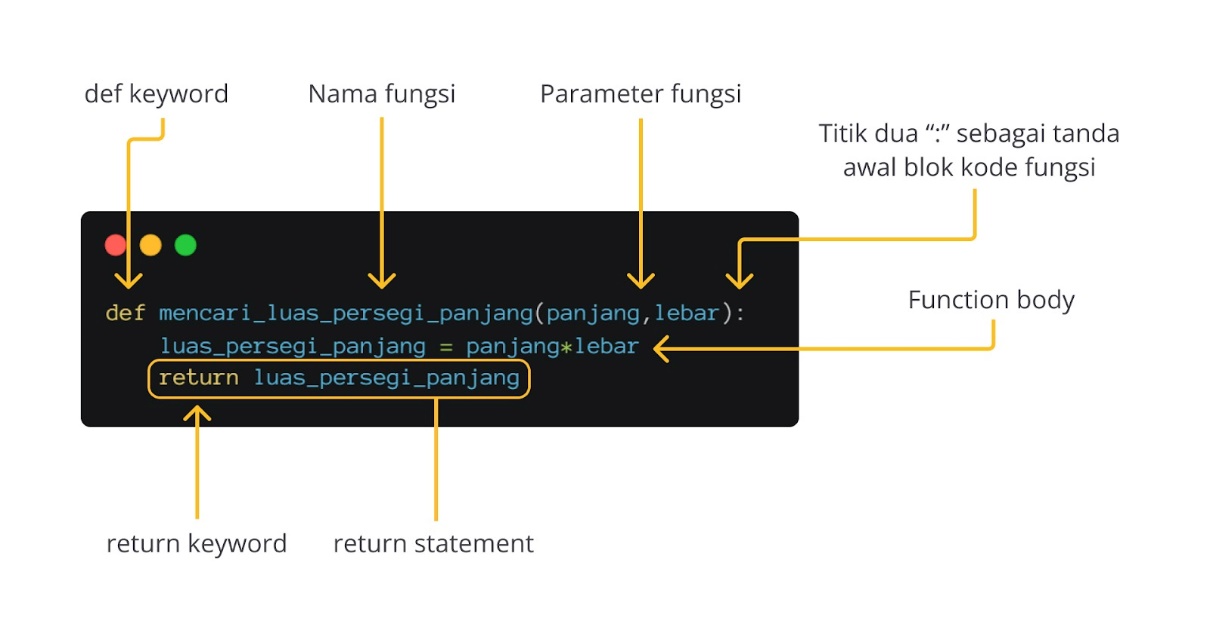
Secara umum, fungsi terdiri dari **header**, **body**, dan **return**,seperti gambar berikut.



Dengan catatan sebagai berikut.

1. *Function header*memberi tahu Python bahwa kita mulai mendefinisikan suatu fungsi.
2. *Function body* adalah blok kode yang diindentasi setelah header fungsi menentukan hal yang dilakukan fungsi tersebut. Kita menentukan semua kode yang akan dibuat dalam *function body*. Ingat bahwa bagian ini adalah blok kode sehingga Anda harus memperhatikan indentasi untuk menghindari kesalahan.
3. *Function return*adalah pernyataan yang digunakan dalam fungsi untuk mengembalikan nilai atau hasil eksekusi dari fungsi tersebut. Ketika sebuah fungsi dieksekusi, biasanya ada situasi bahwa kita ingin mendapatkan nilai atau hasil dari proses yang dilakukan oleh fungsi tersebut. Untuk itu, kita menggunakan pernyataan 'return' dalam fungsi untuk mengembalikan nilai kepada pemanggil fungsi.

Ingat bahwa permasalahan yang ingin diselesaikan adalah kita tidak ingin mengulang untuk menulis rumus luas persegi panjang. Jadi, kita menggunakan fungsi untuk membungkus rumus tersebut. Perhatikan gambar di bawah ini untuk lebih menjelaskan setiap elemen fungsi.



Fungsi yang kita buat sebelumnya terdiri dari beberapa elemen, yakni berikut.

1. **def** digunakan sebagai pernyataan bahwa kita membuat fungsi.
2. "**mencari\_luas\_persegi\_panjang**" merupakan nama fungsi yang kita tentukan.
3. Setiap fungsi memiliki **parameter**bertujuan untuk menyimpan nilai yang akan digunakan oleh fungsi dalam proses eksekusinya. Dalam contoh mencari luas persegi panjang, parameter "panjang, lebar" akan menyimpan setiap input yang masuk, seperti jika Anda memasukkan panjang 5cm dan lebar 10cm.
4. Setiap definisi fungsi harus diakhiri dengan ***colon* atau titik dua ":"**untuk menandakan awal blok kode fungsi.
5. Setelah **function header**selesai, selanjutnya kita definisikan **function body** yang berisi kode untuk dieksekusi. Dalam contoh mencari luas persegi panjang, kita memasukkan rumus luas persegi di bagian ini. Hasil dari rumus tersebut disimpan dalam variabel "luas\_persegi\_panjang".
6. Terakhir, kita menggunakan **return keyword**yang merupakan bagian dari **return statement**. Return statement bertujuan untuk mengembalikan nilai atau hasil eksekusi fungsi tersebut. Dalam contoh di atas, kita mengembalikan variabel "luas\_persegi\_panjang" sebagai hasil dari proses eksekusi fungsi.

Memanggil Fungsi

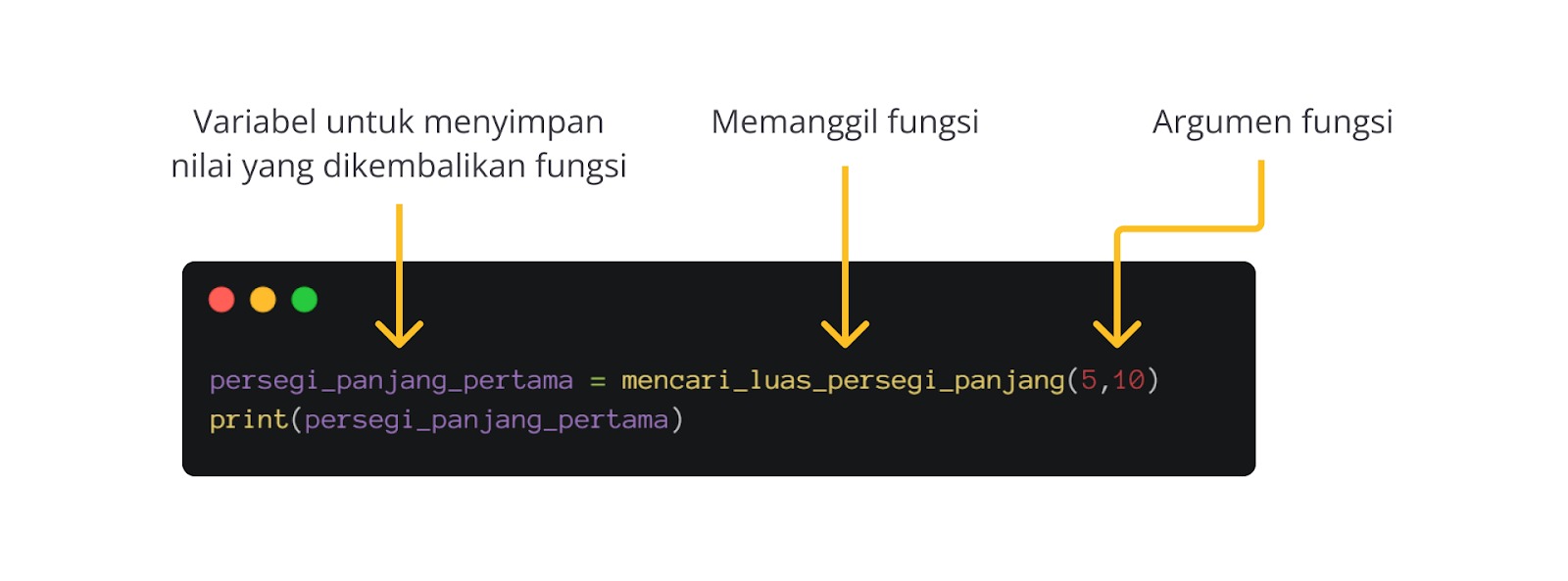
Sekarang, bagaimana cara memanggil fungsi tersebut? Pada dasarnya, kita bisa memanggil fungsi dengan cara menuliskan nama fungsi seperti kita menggunakan "print()", "len()", dan sebagainya.

|  |
| --- |
| def mencari\_luas\_persegi\_panjang(panjang,lebar):  luas\_persegi\_panjang = panjang\*lebar  return luas\_persegi\_panjang  persegi\_panjang\_pertama = mencari\_luas\_persegi\_panjang(5,10)  print(persegi\_panjang\_pertama) |

Dari kode di atas, secara sekuensial program akan berjalan seperti berikut.

1. Pertama, kita menggunakan fungsi bernama "**mencari\_luas\_persegi\_panjang()**" yang memiliki dua parameter, yaitu panjang dan lebar. Fungsi ini menghitung luas persegi panjang dengan mengalikan nilai panjang dan lebar serta mengembalikan hasilnya.
2. Ketika memanggil fungsi "mencari\_luas\_persegi\_panjang(5, 10)", kita menyimpan hasilnya dalam variabel bernama "**persegi\_panjang\_pertama**". Dalam pemanggilan ini, angka 5 dianggap sebagai nilai panjang dan angka 10 dianggap sebagai nilai lebar.
3. Kemudian, dalam fungsi, nilai panjang dan lebar digunakan untuk menjalankan kode "**panjang \* lebar"**. Hasil dari operasi ini disimpan dalam variabel "luas\_persegi\_panjang".
4. Selanjutnya, kita mengembalikan nilai luas\_persegi\_panjang kepada pemanggil fungsi dengan menggunakan pernyataan "return".
5. Sekarang, nilai yang dikembalikan oleh fungsi tersebut disimpan dalam variabel "persegi\_panjang\_pertama" sehingga kita dapat mencetaknya ke layar menggunakan fungsi "print(persegi\_panjang\_pertama)".

Ke depannya, kita bisa menggunakan variabel tersebut berulang-ulang. Secara struktur, pemanggil fungsi terdiri beberapa elemen, yakni berikut.



Ketika Anda memanggil sebuah fungsi, ada dua elemen sebagai berikut.

1. Nama fungsi; tentu Anda harus menyebutkan nama fungsi yang ingin digunakan. Namun ingat, gunakan kurung tutup "()" untuk memanggilnya.
2. Argumen bisa dikatakan sebagai nilai yang diberikan kepada fungsi. Nantinya, nilai tersebut akan disimpan dalam parameter fungsi. Pada contoh fungsi di atas, argumen 5 dan 10 merepresentasikan parameter panjang dan lebar dalam fungsi "mencari\_luas\_persegi\_panjang()" yang kita buat sebelumnya.

**Docstring**

Terakhir, untuk membuat fungsi lebih mudah dipahami oleh programmer lain, kita bisa membuat **dokumentasi berupa *docstring***. *Docstring*adalah akronim dari *documentation string*, bertujuan untuk membuat dokumentasi terhadap fungsi yang dibuat. Umumnya, dokumentasi yang dijelaskan berupa argumen, return, deskripsi fungsi, dan sebagainya.

|  |
| --- |
| def mencari\_luas\_persegi\_panjang(panjang,lebar):  """  Fungsi ini digunakan untuk menghitung luas persegi panjang.  Args:  panjang (int): Panjang persegi panjang.  lebar (int): Lebar persegi panjang.  Returns:  int: Luas persegi panjang hasil perhitungan.  """  luas\_persegi\_panjang = panjang\*lebar  return luas\_persegi\_panjang  persegi\_panjang\_pertama = mencari\_luas\_persegi\_panjang(5,10)  print(persegi\_panjang\_pertama) |

Pada kode di atas, kita mendefinisikan docstring dengan memberikan blok komentar dengan tiga double quote (""") tepat di bawah "def" keyword. Hal ini untuk menegaskan sebelum kode dibaca, kita harus memahami dokumentasi di bawahnya terlebih dahulu.

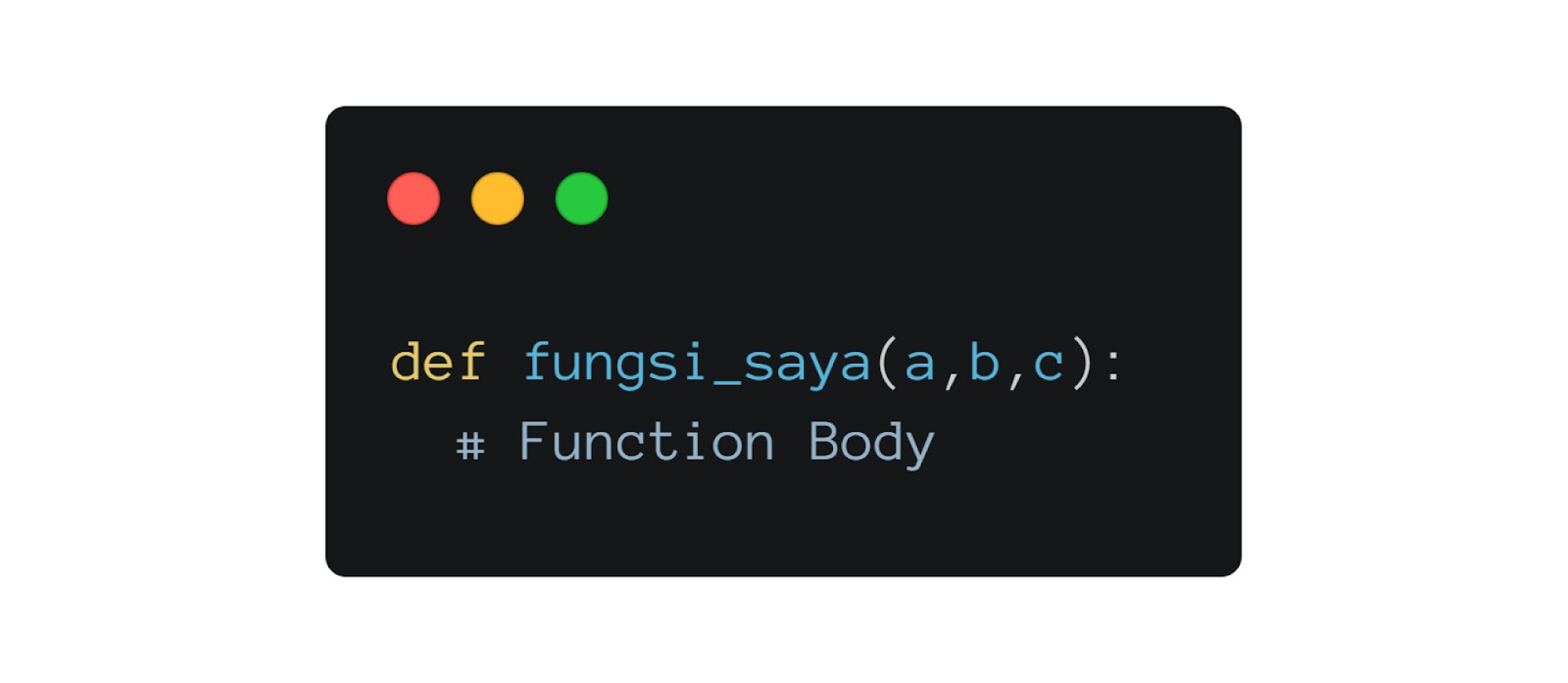
Dokumentasi di atas memiliki tiga elemen, yakni berikut.

1. Deskripsi: Teks yang menjelaskan tujuan dari fungsi yang dibuat. Pada contoh di atas, kita mendefinisikan teks "Fungsi ini digunakan untuk menghitung luas persegi panjang" yang artinya fungsi ini ditujukan untuk menghitung luas persegi panjang.
2. Argumen: bagian yang menjelaskan argumen yang diterima oleh fungsi. Dalam contoh di atas, argumen yang diterima adalah panjang dan lebar dengan keduanya termasuk bilangan bulat atau bertipe data integer.
3. Return: Bagian ini menjelaskan nilai yang akan dikembalikan oleh fungsi. Dalam contoh di atas, fungsi akan mengembalikan nilai luas persegi panjang hasil perhitungan yang termasuk bilangan bulat atau bertipe data integer.

**Argumen dan Parameter**

Ingat bahwa argumen dan parameter adalah hal yang berbeda, sering kali programmer tertukar akan kedua istilah tersebut. Sederhananya, Anda bisa bayangkan **parameter seperti black box** yang akan **menampung nilai**dan **nilai tersebut adalah argumen**.

Contohnya, saat Anda membuat fungsi berikut.



Ketika Anda membuat fungsi di atas, **a, b, c**merupakan parameter. Namun pada saat Anda memanggilnya, **nilai 1, b=50,  c='Dicoding'** menjadi argumen.



**Argumen**

Argumen adalah nilai yang akan diberikan kepada fungsi. Setidaknya ada dua jenis argumen yang dikenal dalam Python.

1. *Keyword Argument*  
   *Keyword Argument* adalah jenis argumen yang disertai dengan nama parameter (*identifier*) dan secara eksplisit disebutkan. Ketika nama parameter dalam sebuah argumen secara langsung disebutkan, artinya kita menggunakan keyword argument.

|  |
| --- |
| def mencari\_luas\_persegi\_panjang(panjang,lebar):  luas\_persegi\_panjang = panjang\*lebar  return luas\_persegi\_panjang  persegi\_panjang\_pertama = mencari\_luas\_persegi\_panjang(panjang=5, lebar=10) |

Ketika kita memanggil fungsi "mencari\_luas\_persegi\_panjang" dengan menuliskan nama parameter diikuti tanda sama dengan (=) dan nilai yang ingin diberikan, itu disebut **keyword argument**. Pada contoh di atas, keyword argument "panjang=5" dan "lebar=10" diberikan saat pemanggilan fungsi. Kelebihan dari jenis argumen ini adalah walaupun kita harus menuliskan lebih banyak kata, urutan parameter fungsi tidak perlu dipikirkan.

1. *Positional Argument*  
   Kebalikan dari *keyword* adalah *positional*, artinya Anda tidak menyebutkan nama parameter (*identifier*) secara eksplisit. Ketika memanggil fungsi, Anda hanya harus memasukkan nilai yang ingin diberikan. Namun, Anda harus mengikuti urutan dari parameter fungsi tersebut.

**Parameter**

Menurut [dokumentasi resmi](https://docs.python.org/3/glossary.html" \l "term-parameter" \t "_blank) Python, ada 5 jenis parameter yang bisa kita atur.

1. *Positional-or-Keyword*  
   Jenis ini merupakan parameter default dalam Python. Dengan jenis ini, kita dapat menggunakan *positional*maupun *keyword argument*ketika memanggil fungsi.

|  |
| --- |
| def greeting(nama, pesan):  return "Halo, " + nama + "! " + pesan  print(greeting("Dicoding", "Selamat pagi!"))  print(greeting(pesan="Selamat sore!", nama="Dicoding"))  Output:  Halo, Dicoding! Selamat pagi!  Halo, Dicoding! Selamat sore! |

Pada contoh di atas, kita membuat fungsi untuk menyapa dengan parameternya adalah “**nama” dan “pesan”**. Ketika memanggil fungsi tersebut, kita bisa menggunakan dua jenis argumen, yakni **positional** dan **keyword**.

1. *Positional-Only*  
   Parameter ini hanya dapat dimanfaatkan dengan menggunakan jenis argumen posisi saat pemanggilan fungsi. Parameter ini ditentukan menggunakan sintaks "/".

|  |
| --- |
| def penjumlahan(a, b, /):  return a + b  print(penjumlahan(8, 50))  Output:  58 |

Pada contoh di atas, kita memanggil fungsi yang telah didefinisikan menggunakan positional argument. Perhatikan juga bahwa kita mendefinisikan parameter fungsi dengan sintaks "/".

1. *Keyword-Only*  
   Jenis ketiga adalah kebalikan dari yang sebelumnya. Kita harus menggunakan keyword argument untuk memanggil fungsi dengan jenis parameter ini. Ia ditentukan dengan sintaks "\*" (asterisk).

|  |
| --- |
| def greeting(\*, nama, pesan):  return "Halo, " + nama + "! " + pesan  print(greeting(pesan="Selamat sore!",nama="Dicoding"))  Output:  Halo, Dicoding! Selamat sore! |

Pada contoh di atas, kita menggunakan keyword argument untuk memanggil fungsi yang telah dibuat. Perhatikan bahwa kita menggunakan sintaks "\*" untuk mendefinisikan bahwa parameter fungsi ini hanya bisa dipanggil menggunakan keyword argument.

1. *Var-Positional* (*Variadic Positional Parameter*)  
   Parameter ini menampung jumlah argumen posisi yang bervariasi saat pemanggilan fungsi. Parameter ini ditentukan dengan menggunakan sintaks **\*args**.

|  |
| --- |
| def hitung\_total(\*args):  print(type(args))  total = sum(args)  return total  print(hitung\_total(1, 2, 3))  Output:  <class 'tuple'>  6 |

Pada contoh di atas, parameter **\*args**mengumpulkan semua argumen posisi yang diberikan saat pemanggilan fungsi dan membungkusnya menjadi tuple "args". Dalam situasi ini, Anda bisa memasukkan angka sebanyak apa pun dalam argumen fungsi.

1. *Var-Keyword* (*Variadic Keyword Parameter*)  
   Parameter ini dapat menampung jumlah keyword argument yang bervariasi saat pemanggilan fungsi. Parameter ini ditentukan dengan menggunakan sintaks **\*\*kwargs** yang berperan sebagai dictionary (seperti tipe datanya). Argumen pada pemanggil fungsi akan berperan sebagai **value**dan parameter (identifier) berperan sebagai **key**.

|  |
| --- |
| def cetak\_info(\*\*kwargs):  info = ""  for key, value in kwargs.items():  info += key + ': ' + value + ", "  return info  print(cetak\_info(nama="Dicoding", usia="17", pekerjaan="Python Programmer"))  """  Output:  nama: Dicoding, usia: 17, pekerjaan: Python Programmer,  """ |

Pada contoh di atas, parameter **\*\*kwargs**akan mengumpulkan semua pasangan key-value yang diberikan sebagai keyword argument. Dalam situasi ini, Anda bisa menambahkan parameter dan argumen sejumlah yang diinginkan.

**Fungsi Anonim (Lambda Expression)**

Terakhir, mari kita pelajari cara membuat fungsi tanpa mendeklarasikan **def**. Cara ini dikenal dengan ekspresi lambda yang digunakan untuk membuat fungsi tanpa perlu menyebutkan **def**ketika membuatnya. Anda bisa mengasumsikan fungsi anonim ini sebagai fungsi one-liner. Secara umum, struktur fungsi anonim sebagai berikut.



Kita akan menggunakan ekspresi lambda untuk melakukan operasi layaknya **def**didefinisikan. **Nama fungsi (func)** setara dengan nama variabel yang digunakan untuk menyimpan ekspresi lambda, **args**adalah argumen yang dibutuhkan untuk dioperasikan, dan **ret\_val** merupakan nilai yang kita kembalikan (return).

Perhatikan contoh di bawah ini. Mari kita ubah contoh fungsi mencari luas persegi panjang menjadi fungsi anonim.

|  |
| --- |
| mencari\_luas\_persegi\_panjang = lambda panjang, lebar: panjang\*lebar  print(mencari\_luas\_persegi\_panjang(5,10))  """  Output:  50  """ |

Pada contoh kode di atas, kita membuat fungsi dengan nama "mencari\_luas\_persegi\_panjang" yang menjadi nama variabel. Argumen yang digunakan adalah **panjang dan lebar**, kita mendefinisikan ini tepat setelah pernyataan **lambda**. Lalu, kita menambahkan isi fungsi, yaitu **panjang\*lebar**dan hasilnya merupakan **return value**. Terakhir, pemanggilan pada fungsi anonim sama seperti biasanya.

Isi fungsi dalam lambda dapat Anda ganti menjadi sebuah nilai, alih-alih variabel. Hal ini karena umumnya bertujuan untuk melakukan operasi sederhana dan perlu melibatkan fungsi yang tidak terlalu kompleks hingga perlu menggunakan **def**.

Silakan ganti **panjang\*lebar**dengan nilai integer yang Anda mau untuk mengetahui maksud dari pernyataan di atas.

Sebuah fungsi lambda dapat menerima argumen dalam jumlah berapa pun, tetapi hanya mengembalikan satu nilai ekspresi. Dalam contoh di atas, Anda bisa menambahkan argumen selain panjang dan lebar, tetapi hanya bisa menggunakan satu operasi, yaitu **panjang\*lebar**.

**Variabel Global dan Lokal**

Dalam Python, ada konsep "scope" yang mengatur jangkauan variabel dan fungsi dalam suatu program. Konsep ini lebih dikenal sebagai *scope variable*yang mengacu pada wilayah di dalam program tempat variabel dapat diakses dan digunakan.

Ada dua jenis scope yang umum dijumpai, khususnya ketika Anda membuat fungsi dan program yang lebih kompleks.

1. Variabel Global  
   Suatu variabel yang didefinisikan di luar fungsi atau blok kode apa pun dan dapat diakses dari seluruh bagian program. Mari lihat penerapannya di bawah ini, asumsikan bahwa kita membuat sebuah fungsi penjumlahan dengan satu nilai yang sudah ditetapkan, yaitu 10.

|  |
| --- |
| a = 10  def add(b):  result = a+b  return result  bilanganPertama = add(20)  print(bilanganPertama)  """  Output:  30  """ |

Pada contoh di atas, kita mendeklarasikan **variabel a** dan menginisialisasikannya dengan nilai 10. Pada tahap ini, kita menetapkan bahwa variabel a merupakan **variabel global** dan dapat digunakan pada seluruh bagian program yang kita buat. Selanjutnya, fungsi penjumlahan didefinisikan dan akan menjumlahkan variabel a yang telah dibuat dengan bilangan yang dapat kita tentukan. Pada contoh di atas, kita menambahkan variabel a bernilai 10 dengan nilai 20 dan hasilnya adalah 30.

1. Variabel Lokal  
   Variabel ini didefinisikan dalam suatu fungsi atau blok kode tertentu. Jenis ini hanya dapat diakses dari dalam fungsi atau blok kode tempat variabel tersebut didefinisikan. Mari lihat contohnya di bawah ini, asumsikan bahwa kita membuat fungsi untuk penjumlahan yang menerima dua bilangan untuk dijumlahkan. Namun, kali ini setiap dua bilangan tersebut dioperasikan akan dikurangi oleh lokal variabel bernilai 5.

|  |
| --- |
| def add(a, b):  lokal\_var = 5  result = a+b-lokal\_var  return result  bilanganPertama = add(10,20)  print(bilanganPertama)  """  Output:  25  """ |

Pada contoh di atas, kita mendefinisikan fungsi penjumlahan bernama "add" dengan dua parameter, yakni a dan b. Dalam fungsi tersebut, kita mengoperasikan penjumlahan antara a dan b lalu dikurangi variabel lokal bernama "lokal\_var" dengan nilai 5.Ingat bahwa variabel lokal hanya dapat diakses dari dalam fungsi atau blok kode tempat variabel tersebut didefinisikan. Silakan coba cetak ke layar "lokal\_var" menggunakan fungsi **print().**Salin dan tempel kode berikut ke dalam program di atas tepat setelah perintah **print(bilanganPertama).**

**Menulis Modul pada Python**

Pembahasan terakhir terkait fungsi adalah kita akan mempelajari cara memanggil sebuah fungsi dari berkas lain. Masih ingat dengan modul? Ia adalah sebuah file berisi kode Python dan di dalamnya terdapat fungsi, kelas, dan sebagainya.

Sebab setiap file berekstensi .py dapat direferensikan sebagai modul, Anda bisa melakukan **impor**file dari satu file ke yang lainnya. Layaknya ketika Anda menggunakan library, modul, dan sebagainya.

Untuk mengikuti materi kali ini, kami rekomendasikan untuk menggunakan IDE,seperti Visual Studio Code atau PyCharm untuk memaksimalkan pembelajaran. Selain itu, kita akan membuat dua buah file. Pastikan kedua file tersebut berada dalam folder atau direktori yang sama. Mari kita mulai.

1. Pertama, buat sebuah file bernama **hello.py**berisi fungsi yang ingin kita impor. Berikut kodenya.
2. Kedua, buat sebuah file bernama **main.py**.Pastikan ia berada dalam satu folder atau direktori dengan file sebelumnya. Pada file main.py, masukkan kode pemanggil fungsi berikut.

|  |
| --- |
| import hello    persegi\_panjang\_pertama = hello.mencari\_luas\_persegi\_panjang(5,10)  print(persegi\_panjang\_pertama) |

Pertama, kita perlu mengimpor file berisi fungsi yang sebelumnya dibuat dengan cara mendefinisikan "import hello". Ingat bahwa hello adalah nama file kita sebelumnya. Baris kedua dan seterusnya adalah cara yang sama dengan yang telah kita pelajari. Namun, perbedaannya adalah kita memanggil fungsi memakai cara "hello.mencari\_luas\_persegi\_panjang(5,10)" dengan hello adalah nama modul atau file yang telah kita impor.  
  
Sekarang, jalankan file main.py pada terminal Anda. Gunakan kode di bawah ini dan pastikan terminal yang Anda buka sedang berada pada direktori yang sama dengan file yang telah dibuat.

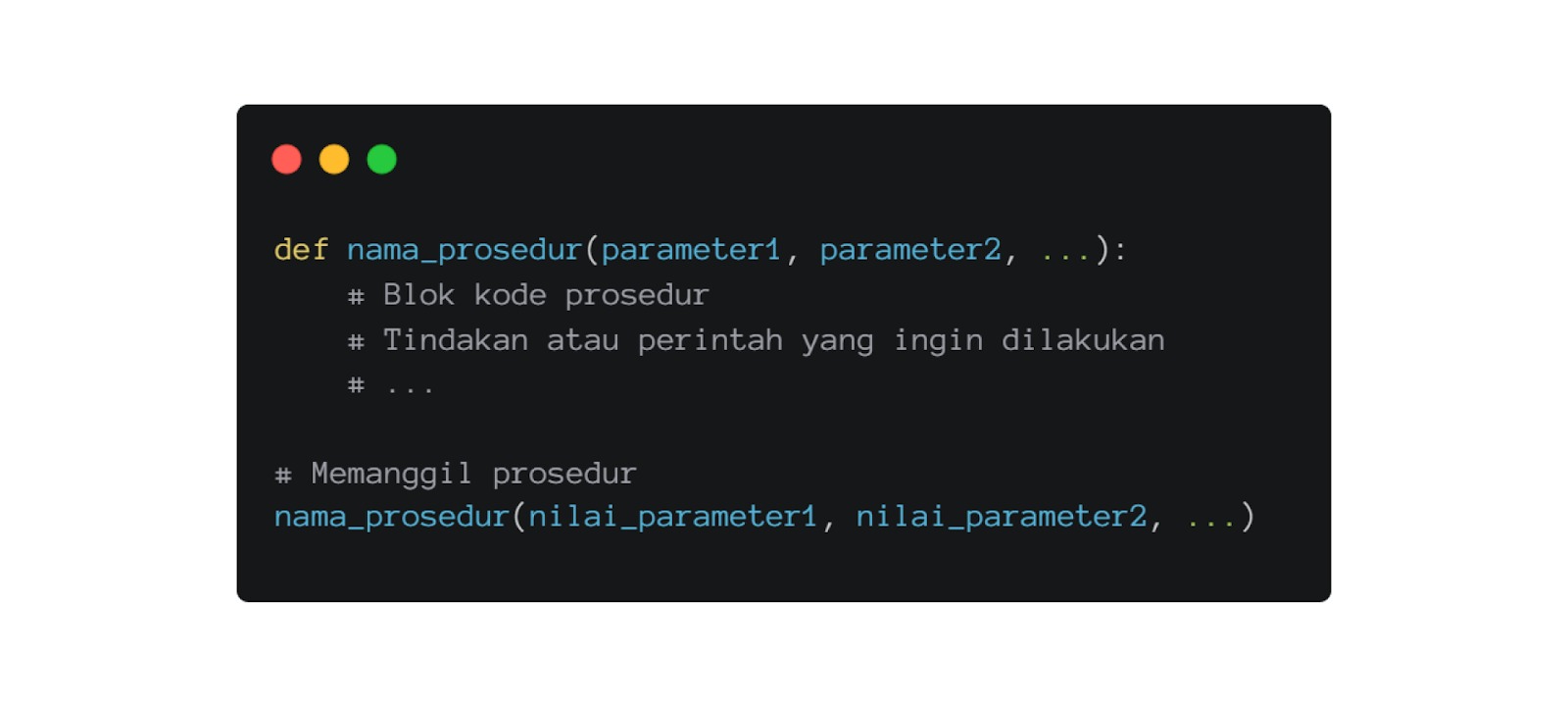
python main.py

**Prosedur**

**Fundamental Prosedur**

Dalam KBBI, kata [prosedur](https://kbbi.kemdikbud.go.id/entri/prosedur" \t "_blank) memiliki makna sebagai tahap kegiatan untuk menyelesaikan suatu aktivitas. Hal ini sama seperti prosedur sebagai subprogram yang merupakan pengelompokan instruksi-instruksi yang sering dipakai dalam program.

Berbeda dengan fungsi, prosedur tidak mengharuskan adanya parameter input atau output dan dapat dipandang sebagai fungsi yang tidak menghasilkan nilai. Dalam Python, prosedur didefinisikan dengan **return tanpa ekspresi atau nilai**yang dihasilkan di akhir fungsi.



Secara konsep, gambar di atas merupakan kerangka dasar prosedur pada Python. Sekilas memang sangat mirip dengan fungsi, hanya saja kita tidak mendefinisikan **return**dan bahkan **return value**.

|  |
| --- |
| def greeting(name):  print("Halo " + name + ", Selamat Datang!") |

Pada contoh di atas, kita membuat fungsi bernama "greeting()" untuk menyapa nama siapa pun yang dikirimkan. Perhatikan bahwa kita tidak mendefinisikan return dan membuat return value. Konsep ini disebut sebagai prosedur, yakni fungsi Python yang kita buat tidak mengeluarkan nilai dari fungsi tersebut.

Kita sebenarnya bisa menambahkan pernyataan **return**, tetapi kita tidak menyertakan **return value** setelahnya.

|  |
| --- |
| def greeting(name):  print("Halo " + name + ", Selamat Datang!")  return |

Pada kode di atas, kita membuat prosedur dengan kode yang sama seperti sebelumnya, hanya saja kita **mendefinisikan return**dan **tidak memberikan return value** setelahnya.

**Mendefinisikan dan Memanggil Prosedur**

Untuk memanggil prosedur, caranya serupa seperti Anda memanggil fungsi. Cukup mendefinisikan satu baris instruksi, seperti "greeting()". Untuk pemberian argumen dan parameter pada prosedur, kita dapat memakai cara yang sama seperti pada fungsi yang telah dijelaskan sebelumnya.

|  |
| --- |
| def greeting(name):  print("Halo " + name + ", Selamat Datang!")  greeting("Dicoding Indonesia")  """  Output:  Halo Dicoding Indonesia, Selamat Datang!  """ |

Pada contoh di atas, kita mendefinisikan prosedur tanpa adanya return statement. Perhatikan bahwa program tetap menampilkan hasil fungsi karena kita langsung menggunakan fungsi "print()" dalam prosedur yang telah dibuat. Jadi, walaupun tidak ada return atau return value, kita tetap mendapatkan output-nya.

Kita juga bisa membuat prosedur yang tidak memiliki parameter input. Ketika kita memanggil prosedur tersebut, program akan langsung menjalankannya. Mari kita ubah kode di atas jika tidak memiliki parameter.

|  |
| --- |
| def greeting():  print("Halo Selamat Datang!")  greeting()  """  Output:  Halo Selamat Datang!  """ |

Pada contoh di atas, prosedur “greeting” tidak menggunakan parameter sama sekali. Namun, sebab body prosedur tersebut memiliki perintah “print()” untuk menampilkan pesan “Halo Selamat Datang!” ke layar, prosedur tersebut akan menampilkan teks ketika kita memanggilnya dengan perintah “greeting()”.