Modul Praktikum
Pengantar Dart dan Flutter

Bagian 1: Pengantar DART

Yudi Wibisono (yudi@upi.edu)
Ilmu Komputer, Universitas Pendidikan Indonesia (cs.upi.edu)

versi: Beta Feb 2024

Daftar Isi

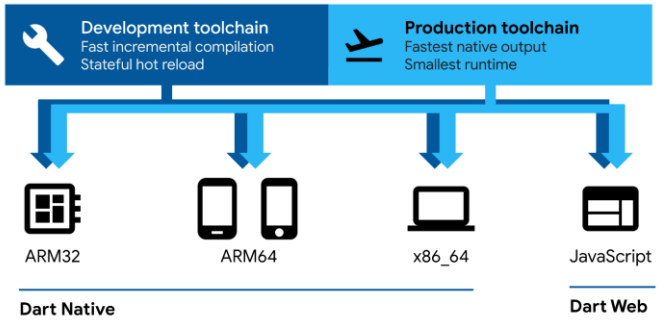
Pendahuluan

Dart adalah bahasa pemrograman yang dikembangkan Google yang awalnya ditujukan untuk pengganti JavaScript. Dart adalah komponen inti dari framework Flutter sehingga untuk menguasai Flutter, kita juga perlu mendalami Dart.

Beberapa karakteristik Dart:

1. Ditujukan untuk multiplatform.
2. Ditujukan untuk pembuatan UI.
3. Mirip seperti Java dengan beberapa penyederhanaan. Ini membuatnya lebih mudah untuk dipelajari.
4. Memiliki kinerja tinggi. Dart menggunakan teknik kompilasi JIT (Just in Time) dan AOT (Ahead of Time). JIT digunakan saat pengembangan app, ditujukan untuk programmer, dioptimasi untuk feedback cepat saat development dan debug. Sedangkan AOT dioptimasi untuk end user karena menghasilkan ukuran code yang kecil (load awal cepat) dan waktu runtime yang optimal. Hanya sedikit bahasa yang mendukung sekaligus JIT dan AOT.
5. Null safety yang mencegah terjadinya null pointer error pada saat runtime karena dicegah saat compile time.
6. Strong type tapi mendukung juga tipe dinamik

Gambar berikut memperlihatkan dua pendekatan Dart saat development dan saat production.



Sumber: https://dart.dev/

Menjalankan Dart

Dart tersedia baik secara online maupun offline.

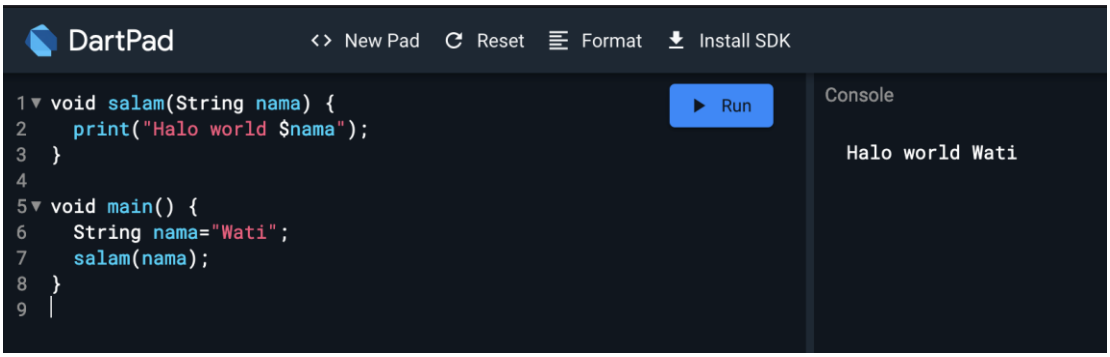
Online

Situs yang dapat digunakan untuk mencoba bahasa Dart secara online https://dartpad.dev/

Berikut contoh contoh code Dart, coba jalankan di situs tersebut.

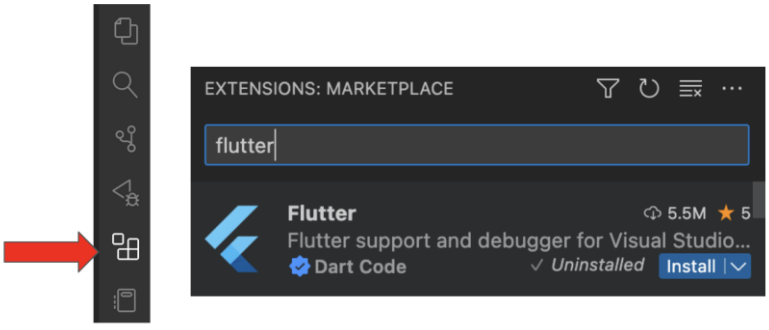
```
void salam(String nama) {
  print("Halo world $nama");
}

void main() {
  String nama="Wati";
  salam(nama);
}
```

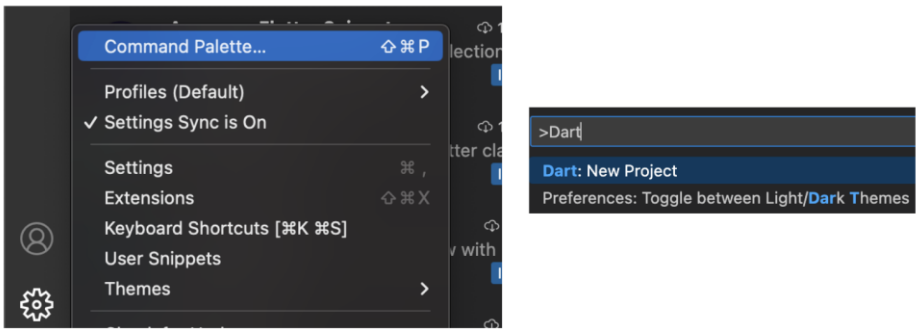


Visual Studio

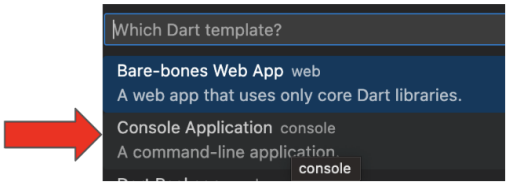
Untuk menginstall Dart pada Visual Studio Code, install extension Flutter



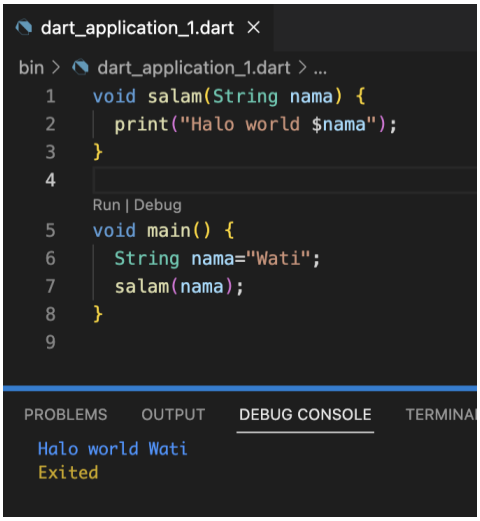
Panggil command Palette, pilih Dart:New Project



Pilih Console Application, ikuti petunjuknya



Jika dijalankan:



Control Flow

Percabangan: if-else

Berikut contoh code untuk if-else statement

```
void main() {
   int nilai = 90;
   if (nilai>=90) {
     print("A");
   } else if (nilai>70) {
     print("B");
   } else {
     print("C");
   }
}
```

Loop: For dan While

for:

```
void main() {
  for (int i = 0; i < 5; i++) {
       print("hello ${i + 1}");
  }
}
```

while-do:

```
void main() {
  int i =1;
  while (i<=5) {
    print(i);
    i++;
  }
}
```

Dart juga mendukung break dan continue. break membuat alur keluar dari loop. Sedangkan continue digunakan untuk melewati (skip) iterasi yang aktif dan lanjut ke iterasi berikutnya.

```
void main() {
  for (int i = 0; i < 5; i++) {
     if (i == 1) {
         continue;
      } else if (i==4) {
         break;
```

```
      }
      print(i);
  }
}
```

Hasilnya:

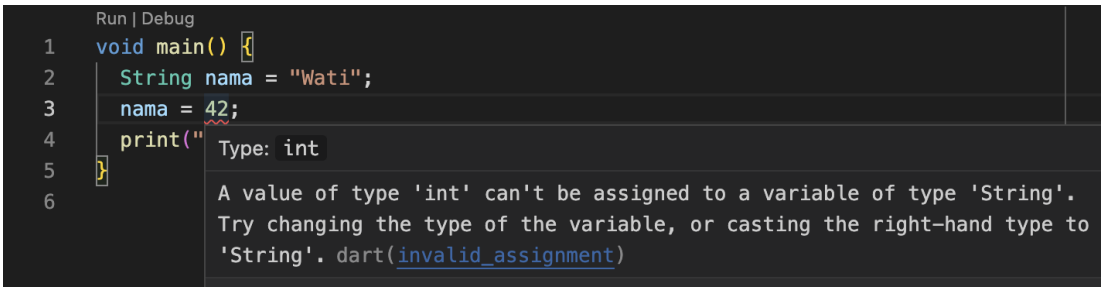
```
0
2
3
```

Tipe Data

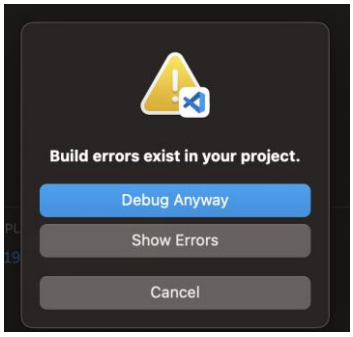
Secara default semua variabel Dart harus dideklarasikan tipenya lebih dulu (type safe).

Program berikut akan memunculkan error pada saat program **belum** dijalankan (compile time error).

```
void main() {
  String nama = "Wati";
  nama = 42; //tipe string tidak bisa diisi int
  print("nama $nama");
}
```



Sangat penting menangkap error sebanyak mungkin pada saat compile time, bukan runtime. Error yang muncul pada runtime dapat menimbulkan kerugian yang jauh lebih besar karena bisa saja muncul di saat tidak terduga.



Tipe-tipe standard Dart adalah int, double, String, bool, list, map dan Symbol.

Contoh:

```
void main() {
  int x = 10;
  double y = 1.2;
  String str = "Wati Martami";
  bool isGagal = false;
  List<int> lAngka = [1, 2];
  Map<String, int> mMhs = {"Budi": 70, "Wati": 60};

  print("$x ; $y ; $str ; $isGagal ; $lAngka ; $mMhs");
}
```

Tipe Var dan Dynamic

Dart menyediakan tipe variabel yang fleksibel yaitu var dan dynamic.

"var" membuat tipe data baru terdefinisi saat di-assign untuk pertama kali. Tetapi setelah di-assign untuk tipe tertentu (misalnya String) maka tidak dapat diubah ke tipe lain. Seperti pada code berikut.

```
void main() {
  var nama = "Wati";
  nama = 42; //ERROR ganti tipe string ke int tidak diperbolehkan
  print("nama $nama");
}
```

Jenis variabel "dynamic" lebih fleksibel dan dapat berubah kapanpun saat runtime.

```
void main() {
  dynamic nama = "Wati";
  nama = 42; //bisa ganti tipe dari string ke int
```

```
  print("nama $nama");
}
```

Jenis variable "var" akan menjadi "dynamic" jika tidak langsung diinisiasi. Tapi ini tidak dianjurkan dan akan mendapat warning.

```
void main() {
  var nama;
  nama = "Wati";
  nama = 42; // bisa
  print("nama $nama");
}
```

Konversi antar Type

parse() dapat digunakan untuk mengkonversi dari tipe string ke integer atau double.

```
void main() {
  String str = '12';
  String str2 = '14.24';
  int num = int.parse(str);
  double num2 = double.parse(str2);
  print(num); // Output: 12
  print(num2); // Output: 14.24
}
```

List

List di dalam Dart mirip dengan List Java, digunakan untuk menyimpan array dinamik. Sebagai contoh:

```
void main() {
  List<int> mylist = [1,2,3];
  mylist.add(4);
  print("Jumlah elemen list: ${mylist.length}");

  //loop semua komponen dalam list, mirip for each
  for (int val in mylist) {
      print(val);
  }

  //cara lain loop elemen
  print("Cara lain loop");
  mylist.forEach((val) {
      print(val);
```

```
  });

  print("Cara lain lebih singkat");
  mylist.forEach((val) => print(val));
}
```

Jika tidak didefinisikan tipenya, secara default List menggunakan tipe Dynamic yang fleksibel.

```
void main() {
  List campuran = [
      1,
      2,
      "Budi",
      true,
      [5, 6]
  ];
  print(campuran);
}
```

Dalam beberapa kasus List bertipe campuran bisa sangat bermanfaat. Di Java misalnya, diperlukan pembuatan class yang kompleks untuk mencapai hal yang sama.

Catatan:
`for (x in list)` lebih dianjurkan dibandingkan `list.foreach()` karena lebih jelas.
`foreach()` bermanfaat untuk tipe iterable yang tidak dapat menggunakan for-in loop seperti Map

todo: anonim objek, create objek dalam list → gunakan objek yang sederhana → sering digunakan di Flutter

Map

Tipe Map digunakan untuk menyimpan pasangan key-value. Key dan value bisa memiliki tipe apapun.

```
Map<String,int> umur = {"rudi": 17, "susi":12};
void main() {
  umur["wati"] = 19;
  umur["iwan"] = 21;
  print(umur);
  print(umur["wati"]);
}
```

Loop sekaligus key-value dapat dilakukan dengan foreach()

```
umur.forEach((key, value) {
      print(key);
      print(value);
})
```

atau

```
umur.forEach((nama, u) => print("Nama : $nama, Umur : $u"));
```

Loop berdasarkan **key**, sekaligus juga value pada map:

```
//loop key
for(String nama in umur.keys) {
    print(nama);
    print(umur[nama]);
}
```

Loop berdasarkan **values** saja:

```
//loop values
for(int u in umur.values)
    print(u);
```

Generic

Generic memungkinkan class, function, method menggunakan berbagai macam jenis tipe (tipe jadi parameter) tetapi tetap menerapkan compile-time type safety. Fitur ini mirip seperti Java.

Contoh penerapan pada list:

```
List myIntList=[];  //list dynamic

void main() {
  myIntList.add("xx");       // string bisa masuk
  myIntList.add(123);        // int bisa masuk juga
  print(myIntList);
}
```

Jika ditambahkan **generic <int>** maka kesalahan tipe dapat dideteksi saat compile

```
List<int> myIntList=[];  //pasti list integer

void main() {
```

```
  myIntList.add("xx");    // <--  gagal saat compile
  myIntList.add(123);
  print(myIntList);
}
```

Sama halnya dengan map

```
Map<String,int> myMap ={}
void main() {
   myMap["Budi"] = 5;
   myMap[2]      = 4;  // <--  gagal saat compile
}
```

Tanpa generic tipe tertentu, maka List dideklarasikan sebagai `List<dynamic>` sehingga semua tipe data dapat masuk. Demikian juga `Map` secara default akan dideklarasikan sebagai `Map<dynamic,dynamic>`

Map() dan Where()

Method Map() dan Where() digunakan untuk memetakan satu list ke list lain dengan cepat.

```
void main() {
  var listMhs = ["Wati", "Budi", "budi"];
  print("List 1:");
  print(listMhs);

  // buat list baru, yang isinya uppercase dari list lama
  print("List 2:");
  var listMhs2 = listMhs.map((mhs) => mhs.toUpperCase()).toList();
  print(listMhs2);

  // buat list baru yang isinya nama yang mengandung "ud"
  print("List 3:");
  var listMhs3 = listMhs.where((mhs) => mhs.contains("ud")).toList();
  print(listMhs3);
}
```

Hasilnya:

```
List 1:
Wati
List 2:
[WATI, BUDI, BUDI]
List 3:
[Budi, budi]
```

Contoh kombinasi where() dan map()

```
List<int> numbers = [1, 2, 3, 4, 5];
List<int> squaredEvens = numbers.where((int x) => x % 2 == 0).map((int x) => x * x).toList();
```

Latihan

- Hasil code berikut adalah:
```
List<int> l1 = [1, 2, 3];
var l2 = l1.map((e) => 2 * e).map((e) => "x$e");
```
- Berdasarkan contoh di atas, gabungkan proses uppercase dan pencarian dalam satu statement. Setelah dijadikan uppercase langsung di-select yang mengandung "UD".
- Buat map yang mengubah isi list menjadi kuadratnya. Misal input adalah [1,2,3] maka hasil pemetaannya adalah [1,4,9]

Spread Operator

List dapat ditambahkan pada list lain secara deklaratif dengan spread operator yang menggunakan tiga titik.

```
void main() {
  List x1 = [1, 2, 3];
  List x2 = [ ...x1, 4];
  List x3 = [0, ...x1];
  List? x4;
  List x5 = [1, ...?x4];  //bisa null
  print(x2);
  print(x3);
  print(x5);
}
```

```
Hasilnya

[1, 2, 3, 4]
[0, 1, 2, 3]
[1]
```

Collection IF & Collection For

Collection-IF dan Collection-For bermanfaat untuk membuat list baru secara efisien dan ringkas berdasarkan list yang sudah ada sebelumnya

Contoh collection if:

```
main() {
  int x = 15;
  List x1 = [1, if (x > 10) 100 else -1,500];
  print (x1);
}
```

Hasilnya

```
[1, 100, 500]
```

Catatan:
Collection if dapat digantikan dengan ternary operator

```
List x1 = [1, x > 10? 100:-1, 500];
```

Contoh collection for:

```
void main() {
  List<int>  x1 = [1, 2, 3];
  List<int>  x2 = [0, for (int x in x1) x];
  print(x2);
}
```

Hasilnya:

```
[0, 1, 2, 3]
```

if dan for dapat dikombinasikan seperti contoh berikut ini

```
  List<int>  x1 = [1, 2, 3];
  List<int>  x2 = [0, for (int x in x1) if (x==2) -1 else x ];
  print(x2);
```

Hasilnya:

```
 [0, 1, -1, 3]
```

Contoh yang lebih kompleks:

```
void main() {
  List<Map<String, dynamic>> pegawai = [
       {"nama": "budi", "hari_lembur":5, "gaji": 10},
       {"nama": "wati", "hari_lembur":2, "gaji": 15},
  ];
  print(pegawai);

  List<Map<String, dynamic>> pendapatan = [
       for (Map<String, dynamic> p in pegawai) {"nama": p["nama"],"penghasilan":
p["gaji"]+p["hari_lembur"]*0.5}
  ];

  print(pendapatan);
}
```

Hasilnya:

```
[{nama: budi, hari_lembur: 5, gaji: 10}, {nama: wati, hari_lembur: 2, gaji: 15}]
[{nama: budi, penghasilan: 12.5}, {nama: wati, penghasilan: 16.0}]
```

Latihan

1. Hasil code berikut adalah:
```
List<int> x1 = [1, 2, 3];
List x2 = [for (int x in x1) "x${x * 2}"];
```

2. Buat code yang berdasarkan input list string, menghasilkan list panjang string.
```
List<String>  s1 = ["satu", "dua", "sebelas"];
```

   Bentuk list output
```
[4, 3, 7]    // 'satu' panjangnya empat dst
```

   Petunjuk: gunakan `str.length` untuk memperoleh panjang string

3. Buat kuadrat dari elemen list yang genap saja

```
input:  [1,2,3,4,5,6];
output: [4, 16, 36]
```

   Petunjuk: gunakan operator % untuk modulo, contoh 4%2 = 0; 3%2 = 1

Const dan Final

Const adalah variabel konstanta yang harus langsung diinisialisasi.

```
void main() {
  const int a=5;
  a = 3; //<-- error karena konstanta tidak boleh diassign
  print(a);
}
```

Variabel final tidak harus langsung diinisialisasi, tapi setelah diinisialisasi tidak dapat diubah.

```
void main() {
  final int a;
  a = 5;
  a = 3; //error karena variabel a sudah final
  print(a);
}
```

Enum

Enum digunakan untuk merepresentasikan kelompok konstanta.

Sebagai contoh:

```
enum jenisMhs {mhsS1, mhsS2, mhsS3}
void main() {
   jenisMhs jm = jenisMhs.mhsS1;
   print(jm);
}
```

Untuk mengakses string dari enum, dapat digunakan .name.
Sebagai contoh:
`jenisMhs.mhsS1.name` akan menghasilkan string `"mhsS1"`

Untuk memetakan dari enum menjadi list string dapat digunakan values kemudian map

```
List<String> namaJenisMhs =
      jenisMhs.values.map((jenisMhs m) {return (m.name);}).toList();
print(namaJenisMhs);
```

Null Safety

Null pointer error (variabel bernilai null yang diakses) sering menjadi masalah dalam aplikasi. Dart menyediakan fasilitas pengamanan terhadap error ini yang disebut null safety sehingga error dapat diketahui saat compile. Berikut contoh pencegahan null pointer error.

```
void main() {
  int x;
  print(x); //error saat compile karena x bernilai null
}
```

Untuk memaksa variabel dapat menerima null, gunakan "?" setelah tipe variabel:

```
void main() {
  int? x;   //tambah simbol ?, artinya nullable
  print(x); //hasilnya tercetak null di layar
}
```

Tetapi compile error akan tetap akan terjadi jika ada **potensi** null pointer error

```
void main() {
  int? x;
  print(x+2);  //walaupun sudah menggunakan ?, tetap error saat compile
}
```

untuk mengatasinya, perlu ditambahkan pengecekan di code

```
void main() {
  int? x;
  if (x != null) {  //harus tambah pengecekan
     print(x + 2);
  }
}
```

atau alternatif yang lebih "berbahaya" adalah dengan menambahkan tanda seru di depan variabel. Ini dilakukan jika kita sangat yakin pada saat variabel digunakan, tidak mengandung null. **Hindari** solusi ini sedapat mungkin, karena rawan error di saat runtime.

```
void main() {
  int? x;
  print(x! + 2);//HATI-HATI tidak akan error saat compile, tapi error saat runtime
}
```

Operator ??= dapat digunakan untuk mengisi nilai default jika nilai variabel kosong.

```
void main() {
  int? x;
  int y = x ??= 0 +2;
  print(y);
}
```

Function & Method

todo:
function
named parameter
anonim objek, create objek dalam method dengan named parameter → gunakan objek yang sederhana → sering digunakan di Flutter

record

```
(String, int) getData() {
  return ("x", 5);
}

void main() {
  String x;
  int y;
  (x,  y) = getData(); //return dua nilai
  print(x);
  print(y);
}
```

Lain-Lain

Arrow Syntax

Arrow syntax (=>) adalah bentuk singkat dari badan fungsi. Seperti pada contoh berikut, bagian kanan arrow adalah ekspresi yang akan dieksekusi.

```
String sayHello(String nama) => "Halo " + nama;

void main() {
  print(sayHello("Wati")); //dipanggil
}
```

Jadi `{ return ekspresi }` dapat disingkat menjadi `=> ekspresi`

Flutter sering menggunakan fungsi anonim bersarang sehingga arrow syntax bermanfaat membuat code lebih mudah dibaca.

Dart juga mendukung fungsi menjadi parameter, sebagai contoh:

```
void proses(fungsi) {
  print(fungsi(2)+50);
}

void main() {
  //passing fungsi sebagai parameter
  proses ((x) => 100*x);    //250
  proses ((x) => 100/x);    //100
}
```

Latihan

1. Apa yang dilakukan code ini?
   ```
   x.where((int x) => x % 2 == 0).map((int x) => x * x).toList();
   ```

Conditional Ternary Operator

Operator ini memiliki tiga operand:

```
kondisi ? ekspresiJikaTrue : ekspersiJikaFalse
```

Contoh penggunaan

```
void main() {
  int value = 5;
  print( (value==0) ? "benar" : "salah"  );
}
```

Operator ternary bermanfaat jika dikombinasikan dengan arrow syntax karena pada operator arrow, statement tidak diperbolehkan (termasuk if-else tidak diperbolehkan). Sebagai contoh:

```
void main() {
  proses((x)=>x>2? x*10: x*2); //berapa hasilnya?
}

void proses(fungsi) {
  print(fungsi(5)+50);
}
```

Contoh nested:

```
void main() {
  int age = 15;
  int salary = 60;
  String status;

  status = age >= 18 ? (salary >= 50 ? "Atas" : "Tengah") : "Underage";
  print("Status: $status");
}
```

Latihan

1. Hasil dari code berikut adalah:

```
var x1 = [1, 2, 3];
var x2 = x1.map((mhs) => mhs==2 ? -20 : mhs).toList();
print(x2);
```

2. Berdasarkan list berikut, buat list baru yang menghitung pajak (<10, pajaknya 15%; >=10 pajaknya 20%)

```
List<Map<String, dynamic>> pegawai = [
  {"nama": "budi", "gaji": 5},
  {"nama": "wati", "gaji": 17},
];
```

List hasil

```
List<Map<String, dynamic>> pajak = [
  {"nama": "budi", "gaji": 5, "pajak":0.15},
  {"nama": "wati", "gaji": 17, "pajak":0.20},
];
```