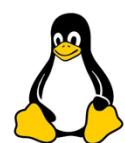
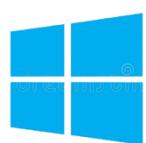
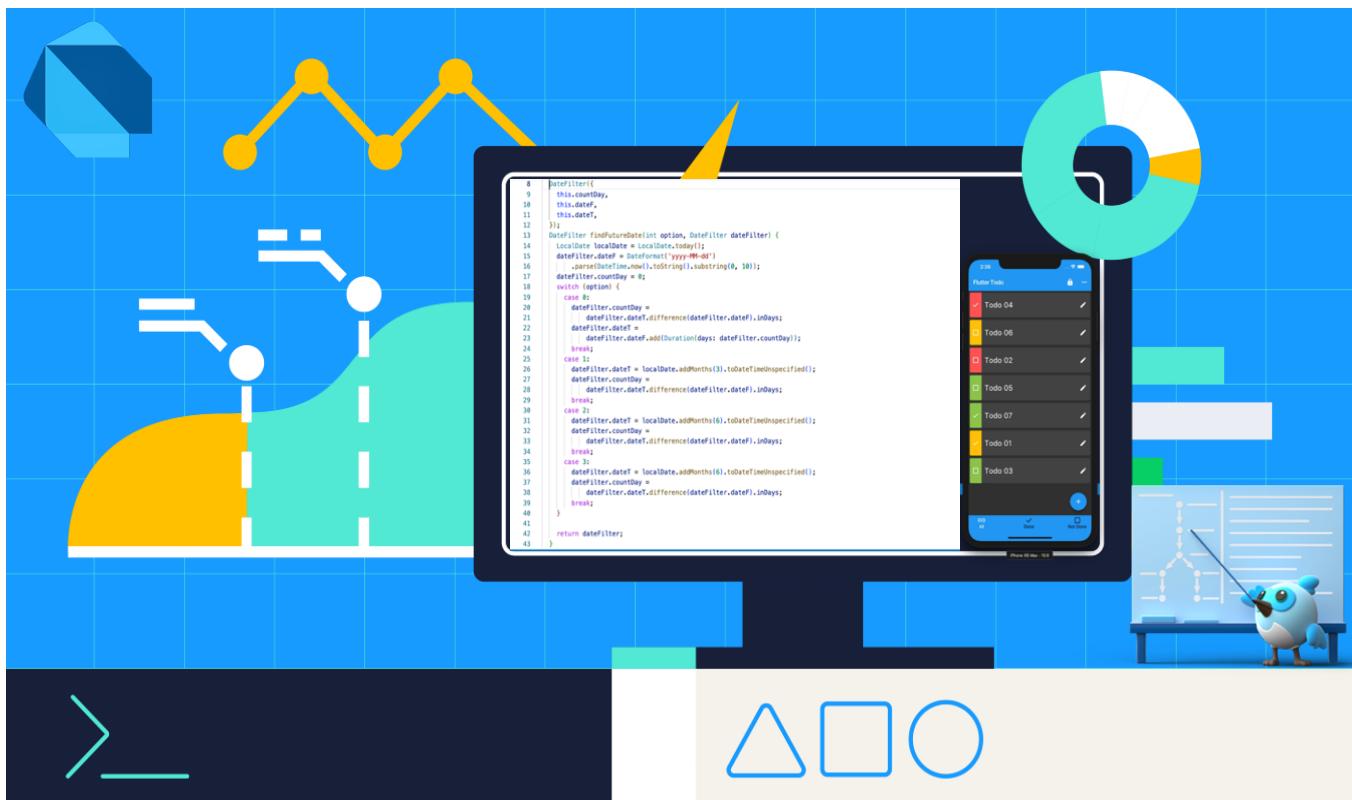




Build Your IT Skill

ETEC IT PROFESSIONAL TRAINING CENTER

Dart Programming



ក្រុមក្រសួងបច្ចេកទេស និង សាស្ត្រ

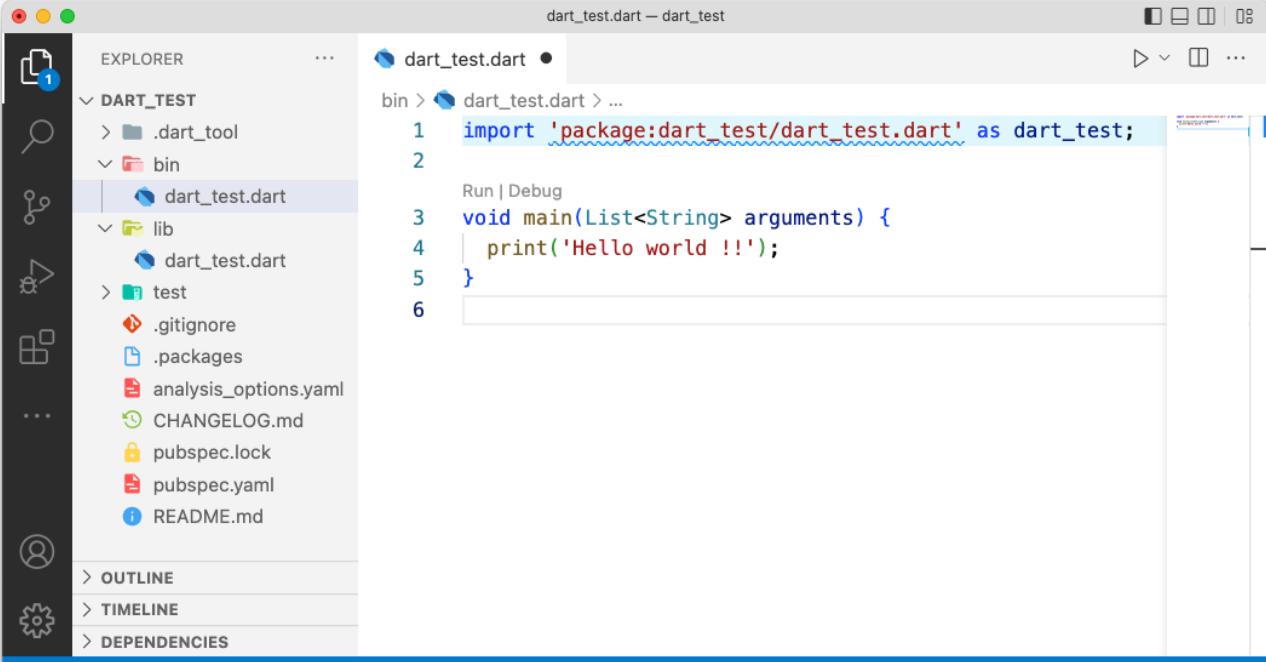
Software Developer

i. Introduction Dart

Dart Programming ត្រូវបានបង្កើតដោយ Google នៅឆ្នាំ 2011

ដែលដោរក្នុងប្រើសម្រាប់សរសេរ Web, service និងក្រុមហ៊្តគេចបាន
Develop នៅលើប្រព័ន្ធសម្រាប់ build

Web , Mobile ,Desktop ជាមួយ Flutter SDK



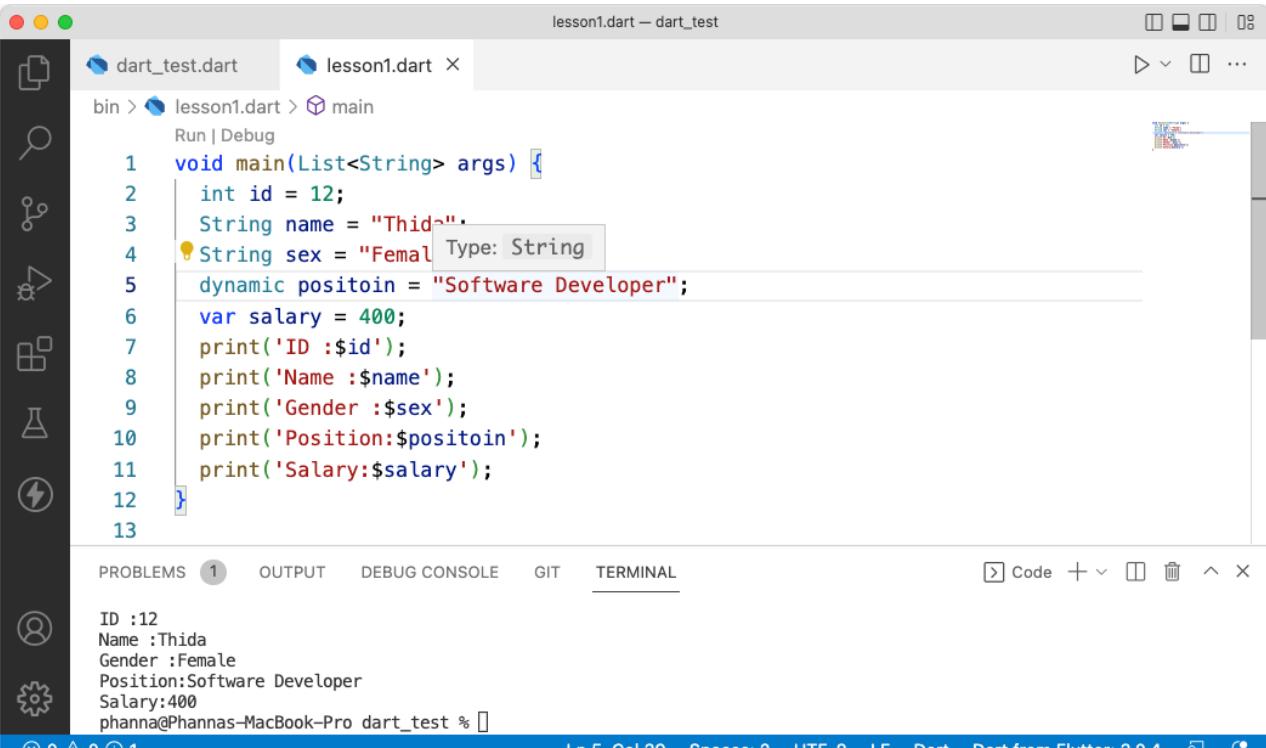
The screenshot shows the Dart IDE interface. On the left is the Explorer sidebar with a tree view of the project structure. The main area displays the code for `dart_test.dart`. The code is as follows:

```
dart_test.dart — dart_test
bin > dart_test.dart > ...
1 import 'package:dart_test/dart_test.dart' as dart_test;
2
3 void main(List<String> arguments) {
4   print('Hello world !!');
5 }
6
```

The status bar at the bottom indicates the file is `ln 6, Col 2`, `Spaces: 2`, `UTF-8`, `LF`, `Dart`, and `Dart from Flutter: 3.0.4`.

ii. Dart I/O

Example 1:



The screenshot shows the Dart IDE interface with two tabs open: `dart_test.dart` and `lesson1.dart`. The `lesson1.dart` tab is active and displays the following code:

```
lesson1.dart — dart_test
bin > lesson1.dart > main
Run | Debug
1 void main(List<String> args) {
2   int id = 12;
3   String name = "Thida";
4   String sex = "Female"; Type: String
5   dynamic positoin = "Software Developer";
6   var salary = 400;
7   print('ID :$id');
8   print('Name :$name');
9   print('Gender :$sex');
10  print('Position:$positoin');
11  print('Salary:$salary');
12 }
```

The terminal below the editor shows the output of the program:

```
ID :12
Name :Thida
Gender :Female
Position:Software Developer
Salary:400
phanna@Phannas-MacBook-Pro dart_test %
```

The status bar at the bottom indicates the file is `ln 5, Col 39`, `Spaces: 2`, `UTF-8`, `LF`, `Dart`, and `Dart from Flutter: 3.0.4`.

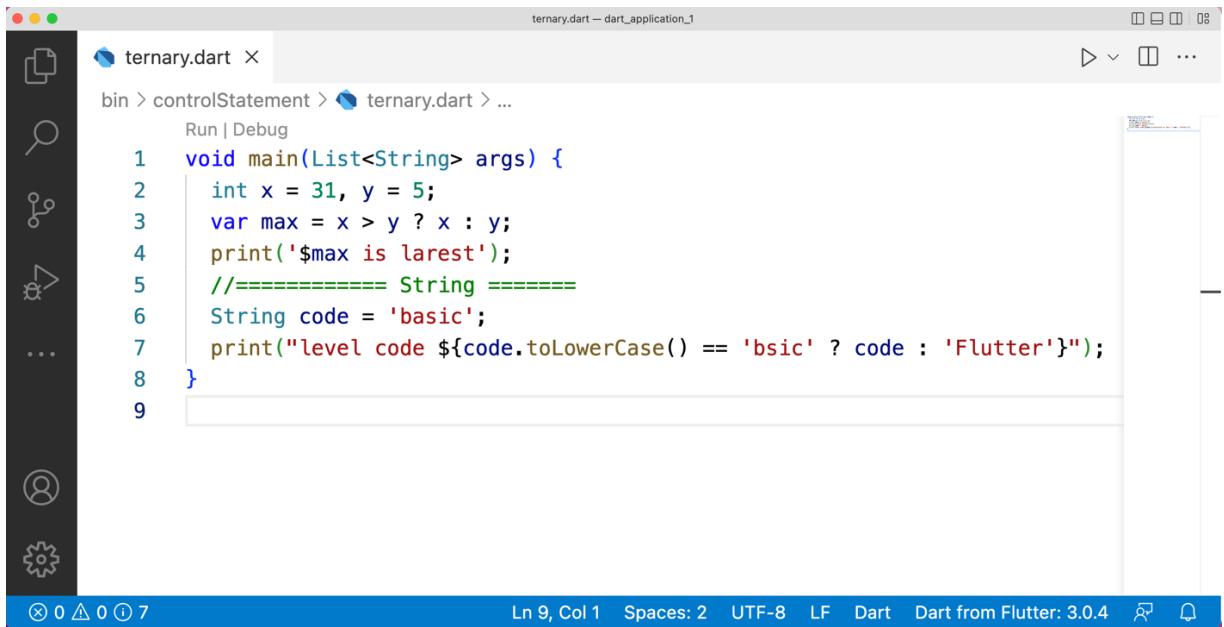
Example2

```
input.dart X
bin > input.dart > main
1 import 'dart:io';
2
3 void main(List<String> args) {
4     int? id;
5     String? name;
6     double? score;
7
8     stdout.write('Input Id:');
9     id = int.parse(stdin.readLineSync()!);
10    stdout.write('Input Name:');
11    name = stdin.readLineSync();
12    stdout.write('Input score:');
13    score = double.parse(stdin.readLineSync()!);
14    print('===== Output =====');
15    print('Id=$id');
16    print('Name=$name');
17    print('Score=$score');
18 }
```

iii. Control Statement

```
bin > controlStatement > ex1.dart > main
Run | Debug
1 void main(List<String> args) {
2     int? x = 54;
3     int? y = 34;
4
5     // ===== ත පිහුණු ======
6     if (x > y) {
7         print('X = $x is largest than Y');
8     }
9     // ===== එ පිහුණු ======
10    if (x > y) {
11        print('X = $x biggest');
12    } else {
13        print('Y=$y biggest');
14    }
15    // ===== ම පිහුණු ======
16    int number = 0;
17    if (number < 0) {
18        print('$number is Negative number');
19    } else if (number > 0) {
20        print('$number is positive number');
21    } else {
22        print('number is $number');
23    }
24 }
```

- Ternary



A screenshot of a Dart code editor window titled "ternary.dart". The code is as follows:

```
void main(List<String> args) {
  int x = 31, y = 5;
  var max = x > y ? x : y;
  print('$max is larest');
  //===== String =====
  String code = 'basic';
  print("level code ${code.toLowerCase()} == 'bsic' ? code : 'Flutter'");
}
```

The editor interface includes a sidebar with icons for file operations, a search bar, and a status bar at the bottom showing "Ln 9, Col 1" and "Dart from Flutter: 3.0.4".

- Switch Statement



A screenshot of a Dart code editor window showing a switch statement. The code is as follows:

```
void main(List<String> args) {
  int codeDay;
  stdout.write('Please Choose 1-7:');
  codeDay = int.parse(stdin.readLineSync()!);
  switch (codeDay) {
    case 1:
      print('Monday');
      break;
    case 2:
      print('Tuesday');
      break;
    case 3:
      print('Wednesday');
      break;
    case 4:
      print('Thursday');
      break;
    case 5:
      print('Friday');
      break;
    case 6:
      print('Saturday');
      break;
    case 7:
      print('Sunday');
      break;
  }
}
```

The editor interface includes a sidebar with icons for file operations, a search bar, and a status bar at the bottom.

iv. Loop statement

Example1:

```
bin > loopStatement > loop1.dart > main
1 import 'dart:io';
2
3 void main(List<String> args) {
4     // ===== forloop =====
5     for (int i = 1; i < 10; i = i + 2) {
6         stdout.write('$i ');
7     }
8     print('\n=====');
9     for (int i = 20; i >= 1; i = i - 2) {
10        stdout.write('$i ');
11    }
12
13     // ===== while loop =====
14     int i = 1;
15     while (i <= 10) {
16         stdout.write('$i ');
17         i = i + 2;
18     }
19
20     // ===== do while loop =====
21     print(' do while loop ');
22     i = 2;
23     do {
24         print(i);
25         i = i + 2;
26     } while (i <= 15);
27 }
```

Example2:

```
bin > loopStatement > loop_foreach.dart > ...
Run | Debug
1 void main() {
2     List names = ['Kim long', 'Chanthu', 'lisa', 'dalin'];
3     for (var temp in names) {
4         print('Name:$temp');
5     }
6 }
7
```

PROBLEMS 7 OUTPUT DEBUG CONSOLE GIT TERMINAL

```
Name:Kim long
Name:Chanthu
Name:lisa
Name:dalin
phanna@Phannas-MacBook-Pro dart_application_1 %
```

Code Code

v. Function

- Non- Return Function

Example1:

```
Run | Debug
1 void main(List<String> args) {
2     sum2(32, 44);
3     sum1();
4 }
5
6 // Non- return function not parameter
7 void sum1() {
8     int x = 34, y = 76;
9     print('X+Y=${x + y}');
10}
11
12 // Non- return function have parameter
13 void sum2(int a, int b) {
14     print('A+B=${a + b}');
15 }
16
```

- Return function

Example2:

```
1 void main(List<String> args) {
2     var x = sum3();
3     print('Sum3=$x');
4     var y = sum4(34, 5);
5     print('Sum4=$y');
6     var sumOfFun = sum3()! + sum4(4, 8);
7     print('Sum of Function=$sumOfFun');
8 }
9
10 int? sum3() {
11     sum4(3, 4);
12     int x = 4, y = 6;
13     return x + y;
14 }
15
16 //return function have Parameter [ int , double , bool ,class ,String ]
17 double sum4(int x, int y) {
18     return (x + y).toDouble();
19 }
```

▪ Function Expressions

```
1 void main(List<String> args) {  
2     printText();  
3     printTextParam('Become to Flutter');  
4     print(printText2());  
5     print(printTextParam2('Next level is Flutter'));  
6 }  
7  
8 void printText() => print('Hello Dart Programming');  
9 void printTextParam(String text) => print(text);  
10 String printText2() => 'Start with Dart';  
11 String printTextParam2(String nextLevel) => nextLevel;  
12
```

PROBLEMS 7 OUTPUT DEBUG CONSOLE GIT TERMINAL

```
Hello Dart Programming  
Become to Flutter  
Start with Dart  
Next level is Flutter  
phanna@Phannas-MacBook-Pro dart_application_1 % █
```

▪ Optional Parameter

```
1 void main(List<String> args) {  
2     student(43, 'Chhanon');  
3     print('=====');  
4     student(3, 'KimSeang', sex: 'Female');  
5 }  
6  
7 void student(int id, String name, {String? sex = 'male'}) {  
8     print('ID=$id');  
9     print('Name=$name');  
10    print('Sex=$sex');  
11 }
```

PROBLEMS 7 OUTPUT DEBUG CONSOLE GIT TERMINAL

```
ID=43  
Name=Chhanon  
Sex=male  
=====  
ID=3  
Name=KimSeang  
Sex=Female  
phanna@Phannas-MacBook-Pro dart_application_1 % █
```

vi. Collection

i. List

```
bin > collection > ex1.dart > ...
Run | Debug
1 void main(List<String> args) {
2     // syntax 1
3     List list = [12, 54, 634, 66, 34, 7, 'Animal'];
4     print(list);
5     // or
6     for (var temp in list) {
7         print('Value=$temp');
8     }
9     // syntax 2
10    List<int> listInt = [23, 555];
11    print(listInt);
12
13    List<String> name = ['channa', 'nary', 'Davuth'];
14    for (var names in name) {
15        print('name=$names');
16    }
17 }
```

Collection,property & Method :

length	shuffle(...)
reversed	singleWhere(...)
hashCode	skip(...)
runtimeType	skipWhile(...)
first	sort(...)
isEmpty	sublist(...)
isNotEmpty	take(...)
iterator	takeWhile(...)
last	toList(...)
single	toSet()
add(...)	toString()
addAll(...)	where(...)
lastWhere(...)	any(...)
map(...)	asMap()
reduce(...)	cast()
remove(...)	clear()
removeAt(...)	contains(...)
removeLast()	elementAt(...)
removeRange(...)	every(...)
removeWhere(...)	expand(...)
replaceRange(...)	fillRange(...)
retainWhere(...)	
setAll(...)	
setRange(...)	

Method & property ដែលធ្វើបច្ចើនមានដូចជា :

- property

length , first ,last ,isEmpty ,isNotEmpty

- Method

.add(...) : add ធាតុចូលមកជំនួយ

.addAll(...) : add ធាតុចូលមកជំនួយសរុប

.firstWhere(...) : filter រកធាតុដែលដូចត្រូវ និង យកដំបូងគេ

.map(...) : បំលែងធាតុទៅជា map ដែលប្រើ Key and Value

.remove(...) : លើបធាតុណាមួយចេញពី list

.sort(...) : តម្រូវបានរៀបចំ

.toString(...) : បំលែង data ទៅជា string

.where(...) : រកធាតុណាមួយ

.every(...) : យកធាតុទាំងអស់ដែលមានដូចត្រូវ

.indexWhere(...) : ចាប់យក index ធាតុទាំងអស់ដែលមានដូចត្រូវ

.contains(...) : យកធាតុទាំងអស់ដែលមានត្រូវក្សាបានដូចត្រូវ

.clear(...) : លើបធាតុទាំងអស់ចេញពី list

.toList(...) : បំលែងធាតុទៅជា list

,....

ii. Map

Example1:

```
1 void main(List<String> args) {  
2     Map map = {'id': 123, 'name': 'Thida', 'score': 500};  
3  
4     print(map);  
5  
6     Map<String, dynamic> map1 = {'id': 123, 'name': 'Thida', 'score': 500};  
7     print(map1);  
8  
9     List<Map<String, dynamic>> list = [  
10        {'id': 123, 'name': 'Thida', 'score': 500},  
11        {'id': 143, 'name': 'Vuthy', 'score': 400}  
12    ];  
13    print(list);  
14    for (var temp in list) {  
15        temp.forEach((key, value) {  
16            print('$key : $value');  
17        });  
18    }  
19 }  
20 }
```

Example2

```
bin > map > test1.dart > ...  
Run | Debug  
1 void main() {  
2     var details = {};  
3     details['Username'] = 'admin';  
4     details['Password'] = 'admin@123';  
5     print(details);  
6 }  
7 |
```

PROBLEMS OUTPUT DEBUG CONSOLE GIT TERMINAL

```
{Username: admin, Password: admin@123}  
phanna@Phannas-MacBook-Pro dart_lesson %
```

Example3

```
bin > map > test2.dart > main
1 void main(List<String> args) {
2     List<Map<String, dynamic>> listMap = [
3         {'name': 'thida'},
4         {'name': 'Sothea'},
5         {'name': 'Mr.Long'},
6         {'name': 'Chanra'},
7         {'name': 'KimSeng'},
8         {'name': 'phanit'}
9     ];
10
11     listMap.removeAt(1);
12
13     for (var element in listMap) {
14         print(element);
15     }
16 }
17
```

PROBLEMS OUTPUT DEBUG CONSOLE GIT TERMINAL

```
{name: thida}
{name: Mr.Long}
{name: Chanra}
{name: KimSeng}
{name: phanit}
phanna@Phannas-MacBook-Pro dart_leasson %
```

Pratise

ចូរយើក Example3 ទៅធ្វើជាម៉ោង ដូចខាងក្រោម ៖

===== Menu =====

1. Add
2. Display
3. Search
4. Update
5. Remove
6. Sort

iii. Collection With Class

Example1:

```
bin > map > student_model.dart > Student
5   class Student {
6     late int id;
7     late String name;
8     late String gender;
9     late double score;
10    Student(
11      required this.id,
12      required this.name,
13      required this.gender,
14      required this.score);
15    void Output() {
16      print('$id\t$name\t$gender\t$score');
17      print('-----');
18    }
19  }
20
Run | Debug
21 void main(List<String> args) {
22   List<Student> stuList = [
23     Student(id: 1001, name: 'Angelika', gender: 'F', score: 70.9),
24     Student(id: 1002, name: 'ARita', gender: 'M', score: 80)
25   ];
26   for (var temp in stuList) {
27     temp.Output();
28   }
29 }
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE GIT TERMINAL

```
1001 Angelika F 70.9
-----
1002 ARita M 80.0
-----
phanna@Phannas-MacBook-Pro dart_lesson %
```

vii. Asynchronous

Asyncronous គឺជារាងនៃការបញ្ចូលព័ត៌មានដែលមិនត្រូវបានបញ្ចូលឡើងទៀត នៅក្នុងការបញ្ចូលព័ត៌មាននេះ គឺជាបញ្ហាដែលបានបញ្ចូលឡើងទៀត។

ជាសម្រេច function នៃគេប្រើប្រើនជាមួយ Local database, cloud data, api service ,...

- Not use asyn & await

The screenshot shows a Dart code editor interface. At the top, there's a navigation bar with tabs for 'PROBLEMS' (containing 1 error), 'OUTPUT', 'DEBUG CONSOLE', 'GIT', and 'TERMINAL'. The main area displays a file named 'bin > map > asyn.dart > main'. The code is as follows:

```
1 Future delayedPrint(int seconds, String msg) {
2   final duration = Duration(seconds: seconds);
3   return Future.delayed(duration).then((value) => msg);
4 }
5
6 main() {
7   print('Life');
8   delayedPrint(3, "Is").then((status) {
9     print(status);
10 });
11 print('Good');
12 }
```

A yellow dot icon is positioned next to the 'main()' keyword, indicating it is the current line of execution. Below the code, the output window shows the following text:

Life
Good
Is

- use `asyn` & `await`

Example1:

```

asyn.dart
bin > map > asyn.dart > main
1 Future delayedPrint(int seconds, String msg) {
2   final duration = Duration(seconds: seconds);
3   return Future.delayed(duration).then((value) => msg);
4 }
5
6 main() async {
7   print('Life');
8   await delayedPrint(3, "Is").then((status) {
9     print(status);
10 });
11 print('Good');
12 }
13

```

PROBLEMS 1 OUTPUT DEBUG CONSOLE GIT TERMINAL

Life
Is
Good
phanna@Phannas-MacBook-Pro dart_lesson %

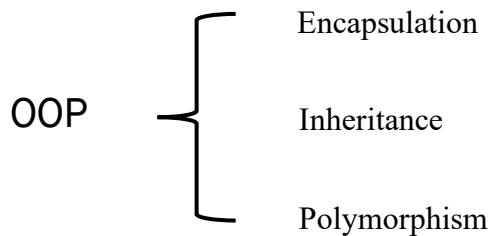
Example2:

```

bin > map > asyn2.dart > ...
1 import 'dart:async';
2
3 Future<String> firstAsync() async {
4   await Future.delayed(const Duration(seconds: 2));
5   return "First!";
6 }
7
8 Future<String> secondAsync() async {
9   await Future.delayed(const Duration(seconds: 2));
10  return "Second!";
11 }
12
13 Future<String> thirdAsync() async {
14   await Future.delayed(const Duration(seconds: 2));
15   return "Third!";
16 }
17
18 void main() async {
19   var f = await firstAsync();
20   print(f);
21   var s = await secondAsync();
22   print(s);
23   var t = await thirdAsync();
24   print(t);
25   print('done');
26 }

```

viii. OOP (Object-Oriented Programming)



i. Encapsulation - Class and Object

```
bin > OOP > class_object.dart > Person > Person
1  class Person {
2    int? id;
3    String? name;
4    String? sex;
5    int? age;
6    Person(
7      required this.id,
8      required this.name,
9      required this.sex,
10     required this.age);
11   Person.newInstance();
12   void output() {
13     print('$id\n$name\n$sex\n$age');
14   }
15 }
16
Run | Debug
17 void main(List<String> args) {
18   Person ps1 = Person.newInstance();
19   ps1.output();
20   print('=====');
21   Person ps2 = Person(id: 12, name: 'Vuthy', sex: 'Male', age: 23);
22   ps2.output();
23 }
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE GIT TERMINAL

```
null null null null
=====
12 Vuthy Male 23
phanna@Phannas-MacBook-Pro dart_leasson %
```

- Property get() , set()

```
1  class Product {  
2      int? code;  
3      String? name;  
4      int? qty;  
5      double? price;  
6      double? total() {  
7          return price! * qty!;  
8      }  
9  
10     void setCode(int code) {  
11         this.code = code;  
12     }  
13  
14     void setName(String name) {  
15         this.name = name;  
16     }  
17  
18     void setPrice(double price) {  
19         this.price = price;  
20     }  
21  
22     void setQty(int qty) {  
23         this.qty = qty;  
24     }  
25  
26     void displayItem() {  
27         print('$code\n$name\n$qty\n$price\n${total()}\n');  
28     }  
29 }  
30  
Run | Debug  
31 void main(List<String> args) {  
32     Product pro = Product();  
33     pro.setCode(1001);  
34     pro.setName('Coca');  
35     pro.setQty(12);  
36     pro.setPrice(1.5);  
37     pro.displayItem();  
38 }
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE GIT TERMINAL

```
1001 Coca 12 1.5$ 18.0$  
phanna@Phannas-MacBook-Pro ~ % dart lesson %
```

ii. Inheritance

- super class

```
class_object.dart ×  
bin > OOP > class_object.dart > main  
1   class Person {  
2     int? id;  
3     String? name;  
4     String? sex;  
5     int? age;  
6     Person(  
7       required this.id,  
8       required this.name,  
9       required this.sex,  
10      required this.age);  
11    Person.newInstance();  
12    void output() {  
13      print('$id\n$name\n$sex\n$age');  
14    }  
15  }
```

- sub class

```
class_object.dart × sudent_model.dart ×  
bin > OOP > sudent_model.dart > ...  
1 import 'class_object.dart';  
2  
3 class Student extends Person {  
4   double? score;  
5   Student({this.score, int? id, String? name, String? sex, int? age})  
6   || : super(id: id, name: name, sex: sex, age: age);  
7   Student.newInstance() : super.newInstance();  
8   @override  
9   void output() {  
10     super.output();  
11     print('Score:$score');  
12   }  
13 }  
14  
15 void main(List<String> args) {  
16   Student stu =  
17   || Student(id: 1002, name: 'Monica', age: 20, sex: 'Female', score: 80.6);  
18   stu.output();  
19 }  
20  
PROBLEMS 1 OUTPUT DEBUG CONSOLE GIT TERMINAL  
1002 Monica Female 20  
Score:80.6  
phanna@Phannas-MacBook-Pro dart_lesson %
```

Interfaces & Abstract class

❖ Interfaces

- Not have key word **interfaces**
- Use key **implements** to implement from super class
- In super class can have data member & function member

Example:

```
implement_and_abstract.dart x
bin > OOP > implement_and_abstract.dart > Item
1  class Product {
2    int? code;
3    String? name;
4    int? qty;
5    double? price;
6    void addProduct() {}
7 }
8
9  class Item implements Product {
10   void updateProduct() {}
11   void showProduct() {
12     print('$code\n$name\n$qty\n$price');
13   }
14
15   @override
16   int? code;
17
18   @override
19   String? name;
20
21   @override
22   double? price;
23
24   @override
25   int? qty;
26
27   @override
28   void addProduct() {}
29 }
```

❖ Abstract

- Use key word **abstract**
- Use key **extent** to implement from super class
- In **abstract class** have abstract function , but function not has body

Example:

```
implement_and_abstract.dart ×
bin > OOP > implement_and_abstract.dart > Item
1 abstract class Product {
2     void addProduct();
3     void updateProduct();
4     void showProduct();
5 }
6
7 class Item extends Product {
8     int? code;
9     String? name;
10    int? qty;
11    double? price;
12
13    @override
14    void addProduct() {
15        // TODO: implement addProduct    TODO: implement addProduct
16    }
17
18    @override
19    void updateProduct() {
20        // TODO: implement updateProduct    TODO: implement updateProduct
21    }
22
23    @override
24    void showProduct() {
25        // TODO: implement showProduct    TODO: implement showProduct
26    }
27 }
28
```

iii. Polymorphism

```
poly.dart  x
bin > OOP > poly.dart > main
1  class Car {
2    void driving() {
3      print("driving car 1");
4    }
5  }
6
7  class Honda extends Car {
8    //override method overrides generic driving method
9    @override
10   void driving() {
11     print("driving car 2");
12     super.driving(); //calls generic driving method
13   }
14 }
15
16 void main() {
17   Honda car1 = Honda();
18   car1.driving();
19 }
20
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE GIT TERMINAL

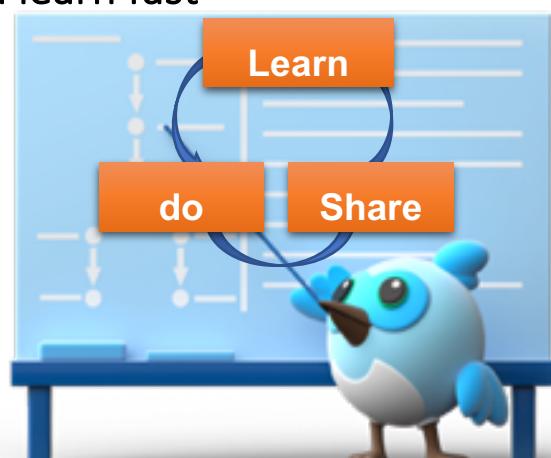
```
driving car 2
driving car 1
phanna@Phannas-MacBook-Pro dart_leasson %
```

Learn more :

Dart: <https://dart.dev/>

Flutter : <https://flutter.dev/>

Habit can make you learn fast



Thank you...!!