



ICT Department

ណែនាំអំពី

Object Oriented Programming ក្នុង Dart

Instructor: Oum Saokosal, Master of Engineering in Information Systems, South Korea '2010

Email: oumsaokosal@gmail.com

Phone: 012 252 752 (Telegram)

- Function គឺជាសំណុំនៃឃ្លាបញ្ជា ដែលមានគោលដៅក្នុងសំរេចការងារអ្វីមួយ។

```
Type functionName (data_type param_1, data_type param_2){  
    return action;  
}
```

- Function អាចសរសេរដោយមិនចាំបាច់ប្រកាស return type ក៏បាន ហើយ datatype របស់ parameter មិនដាក់ក៏បានដែរ។

```
getName(name){  
    return "hello" + name;  
}
```

- យើងអាចប្រើសញ្ញា \Rightarrow សំរាប់ជំនួសអោយកូដ function ខ្លីៗ។
- សញ្ញា \Rightarrow អាចប្រើជំនួសអោយ return ប្រសិនបើមានការបោះតម្លៃទៅអោយ function។

```
Type functionName (data_type param_1, data_type param_2)  $\Rightarrow$  expr;
```

```
bool isPositiveNum(int a)  $\Rightarrow$  a >= 0;
void showSum(int a, int b)  $\Rightarrow$  print("${a+b}");
void main(){
    print(isPositiveNum(-1));    // false
    showSum(5, 7);              //print("12")
}
```

- Function ក្នុងភាសា Dart មានលក្ខណៈពិសេសមួយគឺវា អាចមាន optional parameter (ប៉ារ៉ាម៉ែត្រដាក់ក៏បានមិនដាក់ក៏បាន)។ Optional parameter មាន២ប្រភេទ៖
- positional optional parameter: []

```
showProfile(String name,[int age, String gender]){  
    //do something  
}  
  
showProfile("Sok San");  
showProfile("Sok San", 25);  
showProfile("Sok San", 25, "male");
```

- Naming optional parameter: { }

```
showProfile(String name,{int age, String gender}){  
    //do something  
}  
  
showProfile("Sok San");  
showProfile("Sok San", age: 25);  
showProfile("Sok San", gender: "male", age: 25);
```

- អាចប្រើ = ដើម្បីកម្រៃ default អោយប៉ារ៉ាម៉ែត្រ

```
String showProfile({String name = "vannak", int age = 30, String gender}){  
    print("$name, $age, $gender");  
}
```

```
showProfile(); //vannak, 30, null
```

```
showProfile("Sok San", age: 25); //Sok San, 25, null
```

```
showProfile("Sok San", gender: "male", age: 25); //Sok San, 25, male
```

- **class** គឺជាពុម្ពសំរាប់បង្កើត object
- នៅក្នុង file dart មួយ, យើងអាចដាក់ class បានច្រើន ដោយឈ្មោះ class និងឈ្មោះ file ពុំចាំបាច់ត្រូវមានឈ្មោះដូច Java នោះទេ។
- ក្នុងភាសា dart គឺគ្មាន keyword សំរាប់ public និង private ទេ។ គឺគ្រាន់តែដាក់សញ្ញា underscore ពីមុខឈ្មោះ class គឺក្លាយជា private ហើយ។ បើមិនដាក់ underscore នោះវាគឺជា public។ សំរាប់ protected គឺមិនមានទេក្នុង Dart។

```
class Student{  
    String name;  
    int age;  
}  
  
class _Subject{  
    String title;  
    double duration;  
}
```

- Fields ឬ property គឺជាអញ្ញតិដែលបង្កើតក្នុង class
- Setters និង Getters៖ គឺជាលក្ខណៈកំណត់សិទ្ធក្នុងការបញ្ចូលនិងទទួលតម្លៃរបស់ property។ Setter អាចអោយគេដាក់តម្លៃចូល property បាន។ រីឯ getter អាចអោយគេយកតម្លៃអោយ property បាន។
- Constructors មានតួនាទីសំរាប់កំណត់ទីតាំង memory អោយ object ផង ព្រមទាំងជាកន្លែងសំរាប់ផ្តល់តំលៃដំបូង (default value) អោយក្នុងផង។
- Method៖ ជាធម្មតាវាដូចគ្នានឹង function បេះបិទ ដែលគ្រាន់តែ method វាជាផ្នែកមួយនៃ class ណាមួយជាក់លាក់ រីឯ function គឺមិនស្ថិតនៅក្នុង class ណាមួយទេ។

Instance ឬ object?

```
class Student{  
    int id;  
    String name;  
}
```

```
void main() {  
    Student s1; //s1 គឺជា object ដែលមានតំលៃ null  
    Student(); //បង្កើត instance ព្រមទាំងហៅ constructor របស់ Student()  
    Student s2 = new Student(); //បង្កើត s2 ហើយកំណត់អោយស្មើនឹង instance Student()  
    print("$s1 and $s2");  
}
```


- Setters និង Getters៖ គឺជាលក្ខណៈកំណត់សិទ្ធក្នុងការបញ្ចូលនិងទទួលតម្លៃរបស់ property។ Setter អាចអោយគេដាក់តម្លៃចូល property បាន។ រីឯ getter អាចអោយគេយកតម្លៃអោយ property បាន។
- ឧទា៖ ឈ្មោះម៉ាស៊ីនទូរសព្ទដែលទាញចេញមក គឺអាចមើលបាន តែមិនសរសេរចូលវិញបានទេ ចំនុចនេះគឺគេប្រើ getter (read only)។ ចំពោះ password វិញគឺគេអោយបញ្ចូលបាន តែទាញមកវិញមិនបានទេ ចំនុចគេប្រើ setter (write only)។

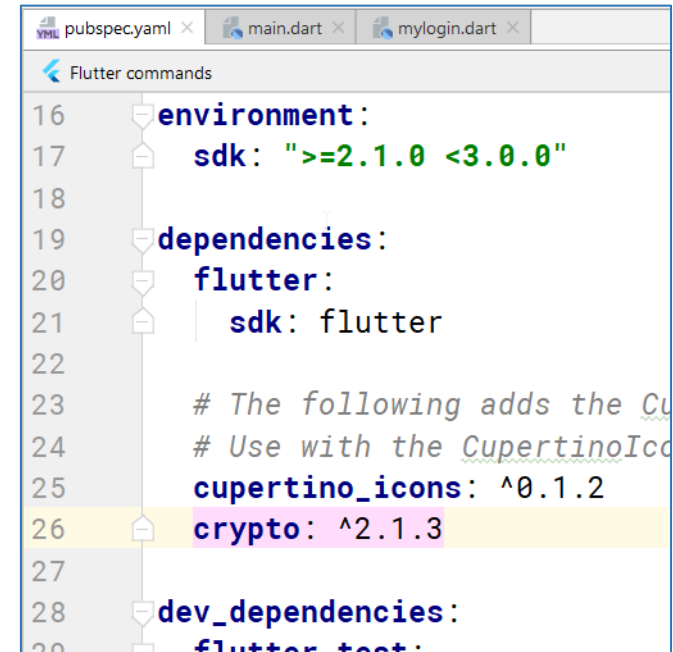
```
void main() {  
    Student s1 = new Student();  
    s1.id = 12;           // Setter  
    s1.name = "dara";     // Setter  
    String n = s1.name;   // Getter  
    print("${s1.id} is ${s1.name}"); // Getter  
}
```

```
import 'package:crypto/crypto.dart';
import 'dart:convert';

class MyLogin{
  String _pass;

  set password(String text){
    var bytes = utf8.encode(text);
    var digest = sha1.convert(bytes);
    _pass = digest.toString();
  }

  String get hashPassword{
    return _pass;
  }
}
```



```
pubspec.yaml
Flutter commands
16 environment:
17   sdk: ">=2.1.0 <3.0.0"
18
19 dependencies:
20   flutter:
21     sdk: flutter
22
23   # The following adds the CupertinoIcons
24   # Use with the CupertinoIcons
25   cupertino_icons: ^0.1.2
26   crypto: ^2.1.3
27
28 dev_dependencies:
29   flutter_test:
```

```
MyLogin login = MyLogin();
login.password = "abc";
String p = login.password; //error on getter
login.hashPassword = "xyz"; //error on setter
String hp = login.hashPassword;
```

- Constructor៖ គឺជា method ពិសេសមួយ ដែលមានតួនាទីសំរាប់កំណត់ទីតាំង memory អោយ object ផង ព្រមទាំងជាកន្លែងសំរាប់ផ្តល់តំលៃដំបូង (default value) អោយកូដផង។
យើងអាចហៅ constructor គឺក្នុងពេលតំណាលគ្នាជាមួយនឹងបង្កើត instance។
 - ❖ Default Constructor
 - ❖ Parameterized Constructor
 - ❖ Named Constructor
- ពេលបង្កើត constructor យើងមិនអាចដាក់ new ក៏បាន មិនដាក់ក៏បាន។

```
Student s1 = new Student();  
var s2 = Student();
```

```
class Student{  
    int id;  
    String name;  
    Student(){  
        id = 10;  
        name = "sok";  
    }  
}
```

```
void main() {  
    Student student1 = Student();  
    print("${student1.id} and ${student1.name}");  
}
```

```
class Student{  
    int id;  
    String name;  
    Student({this.id, this.name = "no name"});  
}
```

```
void main() {  
    Student s1 = Student(); //id = null, name = "no name"  
    Student s2 = Student(id: 1);  
    Student s3 = Student(name: "sok", id: 1);  
}
```

```
class Student{  
    int id;  
    String name;  
    Student(int id, {String name = "no name"}){  
        this.id = id;  
        this.name = name;  
    }  
}
```

```
void main() {  
    Student s1 = Student(10012, name = "Sok San");  
}
```

Named Constructor Constructor មានឈ្មោះ

- នៅក្នុងភាសា Dart, យើងមិនអាចបង្កើត overloading method (method មានឈ្មោះដូចគ្នា) ដូចក្នុងភាសាដទៃបានទេ។ ដូចនេះបើយើងចង់បង្កើត constructor ផ្សេងទៀតគឺយើងត្រូវដាក់ឈ្មោះអោយ constructor នោះ (Named Constructor)។

```
class Student{  
    int id;  
    String name;  
    Student({this.id, this.name});  
    Student.onlyId(int id){  
        this.id = id;  
    }  
}
```

```
void main() {  
    Student student1 = new Student.onlyId(12001);  
}
```



ICT Department

ស៊ីជំរៅលើ
OOP

- Inheritance
- Inheriting Classes and Constructors
- Polymorphism
- Abstract Class
- Interface

- Inheritance ប្រែថា ទទួលមរតក ឬ បកអោយចំន័យ គឺតំណពូជ
- Inheritance គឺជាវិធីសាស្ត្រមួយដែលអនុញ្ញាតិអោយ subclass អាចទទួលបានមរតកដែលមានស្រាប់ពី superclass ណាមួយផ្សេងៗ
- ប្រើ extends keyword
- Subclass អាចមានតែមេតែមួយប៉ុណ្ណោះ

```
class ChildClass extends ParentClass{  
    // class member  
}
```

```
class Vehicle{      // parent class
    String name;
    String model;
    int year;
    Vehicle(this.name,this.model,[this.year]);
}
```

```
class Car extends Vehicle{// sub class
    Car(String name, String model) : super(name, model);
    get getName => this.name;
}
```

Override Method (Method លុប)

- នៅក្នុង subclass យើងអាចសរសេរ override method អោយលុបពីលើ method របស់ class មេបាន។
- ប្រើ @override ពីមុខ method

```
@override  
TypeName methodName() {  
    // TODO: implement  
}
```

```
@override  
String nameVehicle() {  
    // TODO: implement nameVehicle  
    return "this is $branch";  
}
```

- Abstract class ប្រើសំរាប់ធ្វើទំរង់គំរូអោយ class
- ប្រើ abstract keyword ពីខាងមុខ class
- ចំពោះ abstract method គឺមិនមានប្រើ abstract keyword ដូចជាភាសា Java នោះទេ គឺគ្រាន់តែ កុំសរសេរកូដខាងក្នុង ហើយសរសេរវិទ្យុបញ្ចប់ដោយ ; ទៅជាការស្រេច។
- Abstract class មិនអនុញ្ញាតិអោយយើង instance បានទេ
- Abstract class អាចហៅមកប្រើបានត្រឹមតែ datatype តែប៉ុណ្ណោះ
- នៅក្នុង abstract class យើងអាចសរសេរលាយបញ្ចូលគ្នានូវ abstract method និង method ធម្មតាបាន។
- Abstract class គឺជាប្រយោជន៍សំរាប់ polymorphism។

```
abstract class ClassName{  
    // abstract method  
    void showName();  
}
```

```
abstract class Motorcycle{  
    void getName();    //abstract method  
    int numWheel() => 2;  
}
```

```
class Suzuki extends Motorcycle{  
    Suzuki(String motoName) : super(motoName);  
    @override  
    void getName() {  
        print(super.motoName);    // print parent variable  
    }  
}
```

```
void main(){  
    Motorcycle myMoto = new Suzuki("Let");  
    myMoto.getName();  
    print(myMoto.numWheel());  
}
```

- នៅក្នុងភាសា Dart, គេបានបង្កើតអោយ interface មានចំនុចប្រសើរជាភាសាផ្សេងៗដូចជា Java និង C# ជាដើម។ Interface ក្នុង dart គឺអាចបង្កើតឡើង ដោយមិនបាច់ keyword អ្វីពិសេសទេ គឺសូម្បីតែ class សាមញ្ញធម្មតាក៏នឹងអាចក្លាយជា interface របស់ class ផ្សេងបានដែរ។

```
class SimpleClass{
    void show(text) => print(text);
}
class AnotherClass implements SimpleClass{
    @override
    void show(text) => print("here is $t");
}
```

- តែអ្វីដែលសំខាន់គឺ subclass គឺត្រូវ override ឡើងវិញទាំងអស់ទាំង property និង method ដែលមានក្នុង interface។ លើកលែងតែ constructor គឺមិន override ទេ។
- Class មួយអាចមាន interface បានច្រើន ហើយប្រើ implements keyword។

- ប្រើ abstract class ជា interface

```
// interface
abstract class Person{
    String name;
    String setName();
}
```

```
class Student implements Person, OtherInterface{
    @override
    String name;
    @override
    String setName() {
        // TODO: implement setName
    }
}
```




ICT Department

សំណុំធាតុ
(Collection)

- ក្នុង Dart, List គឺមាន index ត្រឹមត្រូវ និងមានប្រវែងច្បាស់លាស់ ហើយចំនួនធាតុរបស់វាអាចរីករួមបាន។

```
List myList = new List();           // dynamic length
myList.add(12);
myList.add('hi');
myList.add(true);
print(myList);
```

- isEmpty() : បោះតម្លៃ true ប្រសិនបើ List ទទេ
- length() : បោះតម្លៃចំនួនសរុបនៃធាតុដែលមានក្នុង List
- add() : បន្ថែមធាតុ
- remove() : លុបធាតុ
- contains() : បោះតម្លៃ true ប្រសិនបើធាតុពិតជាស្ថិតក្នុង List

```
myList.add("dart");                  // insert String dart to list
myList.length                        // 1 length of list
myList.contains("dart")              // true
myList.remove("dart");               // remove dart from list
myList.isEmpty                       // true
```

- យើងអាចកំណត់ datatype អោយ List ដោយដាក់បញ្ចូលនូវ generic type ក្នុងចន្លោះសញ្ញា < >

```
List<type> myList = new List<type>();
```

```
List<String> myList = new List<String>();  
myList.add("Hello");  
myList.add("World");  
print(myList);
```

- Initialization: យើងអាចអោយតំលៃដំបូង ដោយប្រើសញ្ញា []

```
List<int> myList = [];  
myList.add(12);  
myList.add(7);  
print(myList);
```

- `forEach()`: សំរាប់បញ្ចេញទិន្នន័យតាមទំរង់ `for each`
- `insert()`: បញ្ចូលធាតុចូលចន្លោះនៃទីតាំង `index` ណាមួយ
- `removeAt()`: លុបធាតុចំនួនទីតាំង `index` ណាមួយ
- `reversed`: តំរៀបធាតុបញ្ច្រាស់
- `sort`: សំរាប់តំរៀបទិន្នន័យអាចពីតូចទៅធំ ឬពីធំទៅតូច
- `where()`: អាចកំណត់លក្ខខណ្ឌណាមួយសំរាប់ស្វែងរកធាតុក្នុង `List`

```
List<int> numbers = [4, 2, 4, 1 , 5];
numbers.forEach((n)=>print("number: $n"));
numbers.insert(1, 6); print(numbers);           // [4, 6, 2, 4, 1, 5]
numbers.removeAt(0); print(numbers);            // [6, 2, 4, 1, 5]
List<int> list1 = numbers.reversed.toList();
print(list1);                                   // [5, 1, 4, 2, 6]
numbers.sort((a, b) => a.compareTo(b));
print(numbers);                                // [1, 2, 4, 5, 6]
List<int> list2 = numbers.where((x)=> x > 2).toList();
print(list2);                                  // [4, 5, 6]
```

- `map()`: បង្កើត list មួយថ្មីចេញពីការបំណែងធាតុនីមួយៗ
- `List.from()` ឬ `toList()`: បង្កើត list ចេញពី collection ណាមួយ

```
List<int> numInts = [5, 6, 2, 1, 3];  
print(numInts);    //[5, 6, 2, 1, 3]  
List<double> numDoubles = numInts.map((x) => x.toDouble()).toList();  
print(numDoubles); //[5.0, 6.0, 2.0, 1.0, 3.0]  
List<String> numString = List.from(numInts.map((x) => "num $x"));  
print(numString);  //[num 5, num 6, num 2, num 1, num 3]
```

- Map គឺជាសំណុំធាតុលក្ខណៈជាង key/value
- key និង value អាចមាន Datatype ជាប្រភេទអ្វីក៏បាន
- Key ត្រូវតែតម្លៃមិនស្មើ (unique) ហើយមិន null

```
var myMap = new Map(); or var myMap = {}; // define map
```

```
myMap[1] = "10";  
myMap["name"] = "Instinct";  
myMap["isBool"] = true;  
print(myMap);
```

- យើងក៏អាចកំណត់ datatype អោយ key និង value បានដែរ គឺដាក់តាមរយៈ generic type

```
Map<String, String> words = {};  
data["apple"] = "ផ្លែប៉ោម";  
data["ball"] = "បាល់";
```

```
Map<String, dynamic> jsonData = {"id":1,"product":"iMac","price":1150.0};  
jsonData["qty"] = 5;  
jsonData["date"] = DateTime.now().toString();
```

- `addAll()`: បន្ថែមទិន្នន័យចូលក្នុង map
- `putIfAbsent()`: បន្ថែមទិន្នន័យចូលក្នុង map ប្រសិនបើមិនទាន់មាន
- `keys`: បោះតម្លៃ keys ជាតុទាំងអស់ក្នុង map
- `values`: បោះតម្លៃ value ជាតុទាំងអស់ក្នុង map
- `isEmpty`: បោះតម្លៃ true ប្រសិនបើ map គ្មានធាតុ
- `remove()`: លុបធាតុចេញពី map តាមលក្ខខណ្ឌ key
- `removeWhere()`: លុបធាតុចេញ តាមលក្ខខណ្ឌណាមួយ
- `containsKey()`: បោះតម្លៃ true ប្រសិនបើឃើញ key ក្នុង map
- `containsValue()`: បោះតម្លៃ true ប្រសិនបើឃើញ value ក្នុង map
- `clear()`: លុបធាតុទាំងអស់

```
Map<String, dynamic> jsonData = {"id":1,"product":"iMac","price":1150.0};
print(jsonData);           // {id: 1, product: iMac, price: 1150.0}
jsonData.addAll({"qty": 10});
jsonData.putIfAbsent("sold", () => 5);
print(jsonData.keys);      // (id, product, price, qty, sold)
print(jsonData.values);    // (1, iMac, 1150.0, 10, 5)
print(jsonData.isEmpty);   // false
jsonData.remove("qty");
jsonData.removeWhere((k,v) => k == "product" && v == "iMac");
print(jsonData);           // {id: 1, price: 1150.0, sold: 5}
print(jsonData.containsKey("id"));      // true
print(jsonData.containsValue(1150.0));  // true
```