## 

**Dart**

**(Hassan Fulaih)**

**Agenda:**

**1.**[**Variables**](#Variables)

**1.1 syntax**

**1.2 data type**

**2.**[**Input & Output**](#InputOutput)

**3.**[**Conversion**](#Casting)

**4.**[**Comments**](#Comments)

**5.**[**Operations**](#Operations)

**6.**[**Conditions**](#Condition)

**6.1 if**

**6.1.1 if**

**6.1.2 if..else**

**6.1.3 nested if**

**6.2 switch**

**6.3 Expressions**

**7.**[**Loop**](#Loop)

**7.1 for**

**7.2 while**

**7.3 do..while**

**7.4 for statement**

**7.5 forEach**

**7.6 nested loop**

**7.7 infinite loop**

**8.**[**Exception**](#Exceptions)

**9.**[**List (1D)**](#List1D)

**10.**[**List (2D)**](#List2D)

**11.**[**Sets**](#Sets)

**12.**[**Maps**](#Maps)

**13.**[**Number functions**](#NumMethods)

**14.**[**String functions**](#StringMethods)

**15.**[**Functions**](#Functions)

**15.1** [**definition**](#definition)

**15.2** [**calling**](#calling)

**15.3** [**Functions according to data type**](#functionAccordingToDataType)

**15.4** [**Functions according to Parameters**](#functionAccordingToParameters)

**15.5** [**Functions according to required & optional**](#functionAccordingToRequiredAndOptional)

**15.7**

**16.**[**OOP**](#OOP)

**16.1** [**Class**](#Class)

**16.2** [**Object**](#Object)

**16.3** [**Attribute**](#Attribute)

**16.4** [**Methods**](#Methods)

**17.**[**Constructor**](#Constructors)

**17.1** [**default constructor**](#defaultConstructor)

**17.2** [**named constructor**](#NamedConstructor)

**17.3** [**static**](#static)

**17.4** [**this**](#this)

**18.** [**Encapsulation**](#Encapsulation)

**18.1** [**Access modifier**](#AccessModefier)

**18.2** [**set & get**](#Set_Get)

**19.** [**Inheritance**](#Inheritance)

**19.1** [**override**](#override)

**19.2** [**super**](#super)

**19.3** [**super constructor**](#superConstructor)

**20.** [**Abstraction**](#Abstruction)

**20.1** [**Interface**](#Interface)

**20.2** [**Mixin**](#Mixin)

**21.** [**Final & Const**](#FinalConst)

**22.** [**Enum**](#Enum)

**23.** [**Generic**](#Generic)

**24.** [**Asynchronous programming**](#AsyncProgram)

**24.1** [**Future**](#Future)

**24.2** [**Stream**](#Stream)

**25.** [**API**](#API)

**26.** [**Import**](#Import)

**27.** [**Dart Packages**](#Packages)

**28.** [**File System**](#FileSystem)

**29.** [**Null Safety**](#NullSafety)

**30.** [**Required & late**](#RequiredLate)

**Basics:**

**1. V****ariables متغيرات:**

مكان بعمله فالmemory عشان اخزن قيمة جواه

**1.1 Syntax التكوين :**

Data\_type var\_name = value;

او Vata\_type Var\_name;

Var\_name=value;

وممكن اعرف كذا متغير من نفس النوع فنفس السطر

**1.2 data type انواع المتغيرات :**

1.2.1 String: قيم نصية بكتبه بالعلامة ‘ ’او” ” او’’’ ‘’’ الفرق بينهم ان ال’’’ ‘’’ بتفيد لو عايزة اطبع ال" " او ال’ ’ فالنص او لو النص علي كذا سطر وعايزة اطبعهم بنفس الشكل دا

- Note: الstring ممكن يتعمل عليه عمليات حسابية زي الضرب والجمع مثلا الضرب هيكرر الكلمة بنفس عدد الضرب والجمع هيجمع 2 strings علي بعض سواء كان نص ومتغير نوعه string او نصين او غيره وهكذا

1.2.2 Int: ارقام صحيحة

1.2.3 Double: ارقام عشرية

1.2.4 Bool: قيم منطقية true او false

1.2.5dynamic : بتشمل جميع انواع المتغيرات لو انا مش عارفة احدد الdata type اللي هشتغل بيها فهو هيقبل اي valueهحطها للvariable بعدين حتي لو انا غيرت الvalue مش هيعترض

Note:

- في keyword اسمها var هي مش data type ولكن تعتبر زيها بمعني اني اما بكتب var a=’aliaa’; هو بيشوف الvalue اللي اتحطط فالa وبيحدد علي اساسها الdata type فهيلاقي ان الdata type هتكون string فهيعتبر الa انها string

- لو انا جيت اديله value تانية بعد ما عملت assign للvalue اول مرة هيهحصل error لكن لو initialize بس من غير ما احط value واديته كذا value بعدين هيقبلها عادي

**EX:**

String y= '''aliaa"asaad"'''+'sara'\*2;  
int x=10 ,t=5;  
double z=20.5;  
bool a=true;

dynamic **name**= 'Aliaa';  
var e="aliaa";

e=50;*//error*

var b;

b=50;

b="aliaa";

1.2.6 list : لو عايزة احط كذا قيمة فمتغير واقواسها[]

1.2.7set : لو عايزة احط كذا قيمة فمتغير واقواسها {}

1.2.8map : بتبقي علي شكل key : value وبستخدم الkey فاني اوصل لقيمة الvalue واقواسها {}

(هيتشرحوا قدام)

Notes:

- ممكن احدد نوع الداتا اللي جواهم من خلال اني بكتب نوع الداتا ف<> قبل اسم المتغير

- بيقبلوا انواع مختلفة من الداتا مع بعض فنفس الوقت

**EX:**

List<int>al=[1,2,3];  
Set d={'aliaa',3,5.5};  
Map<String,int> g={'aliaa':10,'asaad':50};

**2.Input & Output:**

طرق استقبال قيمة من الuser :stdin.readLineSync()!;

Note:

* القيمة المستقبلة بيكون نوعها string
* عشان احول القيمة من string لقيمة تانية بعمل converting

طرق الطباعة:

1. print() : بتطبع وتنزل سطر
2. stdout.write(): بتطبع ومتنزلش سطر ولو عايزاه ينزل سطر بحط \n جوا علامة نص
3. stdout.writeln():بتطبع وتنزل سطر

**EX:**

String a= stdin.readLineSync()!;  
stdout.writeln(a);  
stdout.write('aliaa');  
stdout.write('aliaa \n');  
print('aliaa');

**Note :**

* لو عايزة اطبع قيمة متغير بحط قبله $ ولو هطبع كذا قيمة مش نص بحطهم في ${}
* لو طبعت متغير بعد ما عرفته وهو ملوش قيمة هيطبعلي null يعني فاضي

**EX:**

stdout.writeln('aliaa $x');  
stdout.write('aliaa ${x+1}');  
print(u);

**3.Casting / conversion التحويل :**

اما يكون عندي variable وعايزة اغير الdata type بتاعته في كذا طريقة :

1. Parse: اما بكون هحول من string لنوع تاني زي double والint لكن مش بتمشي مع معظم الdata type الباقيين

**Syntax: data\_type.parse(variable or value);**

1. toDataType:مش بتمشي مع كل الdata type برده

**Syntax: value or variable.toاسم الداتا تايب اللي عايزه**

**EX:**

String y='aliaa';  
int x=10;  
String t='5';  
var c=x.toString();  
var z=int.*parse*(t);

**4. Comments:**

بعمله عشان واضح سطر معين فالكود :

//: بكتب دي قبل كل comment هعمله

/\* \*/ : لو هكتب كذا سطر comment

**EX:**

*//aliaa  
/\* asaad  
 abd elsalam\*/*

**5.Operations العمليات:**

العمليات اللي بنفذها عالvariables :

5.1 increment & decrement : ++,--

Note: prefix: لو حطيت الزيادة او النقصان قبل الvariable وهنا بينفذ امر الزيادة او النقصان ويغير قيمة الx الاول بعدين يطبع قيمة الx الجديدة (في حالة في كود طباعة هيظهر الفرق)

Postfix: لو حطيت الزيادة او النقصان بعد الvariable وهنا بينفذ الطباعة بانه يطبع قيمة x القديمة بعدين ينفذ الزيادة او النقصان

5.2arithmetic assignment or: -, + ,\* ,/, &, =,~/

5.3 comparation : > ,< ,>= ,<= ,== ,!=,is ,is!

5.4logic : &&,||,!

Note : العلامة ~/ بترجع ناتج القسمة عدد صحيح من غير كسور

**EX:**

var a=10\*50;  
if(a<=1000 && a>=10 || a!=40)  
 a--;  
++a;  
print(a);

**6.Conditions الشرط:**

بستخدمها في حالة وجود شرط واتحقق هينفذ جمل معينة لو لا مش هينفذها :

**6.1 If:**

6.1.1if : في حالة وجود شرط وهينفذ جمل الif فحالة تحققها لو لا مش هتتنفذ

**Syntax: if(condition )**

**{statements}**

Note : لو جمل الشرط هتكون جملة واحدة ممكن مكتبش ال{}

6.1.2if .. else : في حالة وجود شرطين ولو شروط الif متحققتش هينفذ شروط الelse ولو متحققتش مش هينفذ حاجة

**Syntax: if (condition)**

**{statements}**

**else {statements}**

6.1.3 nested if (if .. else if.. else ) : في حالة وجود اكتر من شرطين

**Syntax: if (condition)**

**{statements}**

**else if (condition)**

**{statements}**

**else {statements}**

Note : لو جمل الشرط هتكون جملة واحدة ممكن مكتبش ال{} وممكن فالnested if مكتبش else فالاخر حسب انا محتاجة الكود ازاي

**EX:**

var a=10\*50;  
var b=1;  
if(a<=1000 && a>=10|| a!=40)  
 a--;  
  
if (a>50)  
{b+=2;++a;}  
else a++;  
  
if (a<4000)  
 b++;  
else if (a==4000)  
 a++;  
else a--;  
  
print(a);

**6.2 Switch :**

بديل للif بتقوم علي قيمة متغير معين وكل قيمة بنحسبها case لو القيمة دي ظهرت هنفذ اللي جوا الcase

**Syntax: switch(var){**

**case value1:{statements}**

**case value2:{statements}**

**default:{statments}**

**}**

Note: ممكن احط ال{} بتاعة الcase والdefault فالswitch وممكن اشيلهم

**EX:**

int x=int.*parse*(stdin.readLineSync()!);  
switch(x){  
 case 1 : {print(x);break;}  
 case 2 : {print(x++);break;}  
 case 3 : print(x--);break;  
 default: print(x+2);  
}

**6.3 Expressions :**

6.3.1 **Ternary operator:**

زي if بس مختصرة بحط الشرط قبل ال? ولو اتحقق هيحقق اللي بعد ال? لو متحققش هيحقق اللي بعد ال:

Syntax: condition? True : False;

6.3.2 val1 ?? val2

مقارنة بين قيمتين لو الاولي فيها قيمة هياخد قيمتها لو قيمتها بnull هياخد قيمة التانية

6.3.3 val1 ??= val2

لو القيمة الاولي بnull هياخد قيمة التانية وهيحطها فقيمة التانية

**EX:**

var a=50;  
var b;  
a>30?a++:a--;  
var c = b??a;  
b??=a;  
print(b);  
print(c);

**7. Loops التكرار:**

بستخدمه اما بيكون عندي كود عايزة اكرره كذا مرة في حالة ان الشرط بfalse واما الشرط بيتحقق بنخرد بره اللوب

**7.1 for :**

**Syntax: for(initialization ; condition ; inc or dec)**

**{statements}**

**7.2 while :**

**Syntax: while(condition){**

**Statements;**

**Increment or decrement;}**

**7.3 do..while:**

**Syntax: initialization;**

**do{statements;**

**inc or dec;**

**}while(condition);**

**EX:**

var x = 1;  
while (x < 10) {  
 print(x);  
 x++;  
}  
  
for (var y = 1; y < 10; y++) {  
 print(y);  
}  
  
var c = 1;  
do {  
 print(c);  
 c++;  
} while (c < 10);

**7.4 for statement :**

طريقة مختصرة للfor الفرق بينها وبين العادية انها بتستخدم مع المجموعات زي الlist (هتتشرح قدام) وغيرها

**Syntax: for ( data\_type var\_name in list\_name)**

**{statements}**

**7.5 forEach :**

طريقة مختصرة للfor شبه الfor statement فانها بتستخدم مع الlist واشباهها

**Syntax: list\_name.forEach( (var\_name) {statements} );**

**7.6 Nested loop:**

عبارة عن كذا لوب جوا بعض سواء for او while او غيره

**7.7 Infinite loop:**

لوب تستمر الي ما لا نهاية اني مثلا احط كوندشن مش هيحصل خالص او اني مكتبش الincrement & decrement فيفضل مستمر للاخر او اني اخلي قيمة الcondition بtrue

**EX:**

var x=1;  
while(x<=10){  
 print("hi");  
  
}

Notes:

* في while ممكن احط الزيادة والنقصان او لا واما الشرط بيحصل بيخرج بره اللوب علي طول
* في do while اما الشرط بيتحقق بينفذ جملة مالشرط زيادة ويخرج بره اللوب مش بيخرج علي طول زي while
* فاللوب المتغيرات اللي بتتعرف فيها بتتعرف في نطاق اللوب يعني لو عوزت استخدم المتغير دا بره اللوب مش هيتعرف عليه وهيعمل ايرور
* لو مثلا عندي nested loop وعايزة عند شرط معين اوقف اللوب اللي بره او اللي جوا عني اتحكم فيهم فبدل ما بكتب شرط وبعده break علي طول انا ممكن اسمي كل لوب باسم واما اعوز اوقفها اقوله يوقفها دي بسميها “Control statement”
* الطبيعي اننا بنخلي الvariable بتاع الloop يكون اسمه i اختصار لiteration

**EX:علي الcontrol statement**

loop1:  
for(int x=0; x<10;x++){  
 loop2:  
 for(int j=0; j<10;j++){  
 print(x\*j);  
 if(x==4)  
 break loop1;  
 }  
}

**8. Exceptions:**

بعملها اما اكون عايزة اتجنب error معين بعد ما برن الكود (زي ان الuser يكون مطلوب منه يدخل قيمة int ودخل مكانها string) فبكتب ان لو حصل حاجة بتعمل error معين يظهرلي الmessage معينة بدل الerror فالبرنامج ميجيبش error ويشتغل طبيعي فبعمل handling للerror

**8.1 try..catch:**

لاي نوع error هيظهر

**Syntax: try{**

**Code statements;**

**}**

**catch(val\_name(اي اسم (بيتخزن فيه الايرور اللي ظهر))**

**{**

**الجمل او الحاجات اللي تظهر كبديل للايرور في حالة حدوثه فالtry**

**}**

**8.2 try..on FormatExceprtion:**

للايرورز اللي من نوع formatException فقط يعني اي ايرور تاني هيظهر مش مالنوع دا هيعمل ايرور والبرنامج هيقف

**Syntax: try{**

**Code statements;**

**}**

**on FormatExceprtion**

**{**

**الجمل او الحاجات اللي تظهر كبديل للايرور في حالة حدوثه فالtry**

**}**

**8.3 try .. throw .. catch :**

بستخدمها اما اكون عايز اعمل exception معين عند مرحلة معينة وممكن احدد نوع الexception اللي عايزاه يحصل

**Syntax: try{**

**Code statements;**

**throw e**

**}**

**catch(val\_name(اي اسم (بيتخزن فيه الايرور اللي ظهر))**

**{**

**الجمل او الحاجات اللي تظهر كبديل للايرور في حالة حدوثه فالtry**

**}**

Notes:

* من طرق تكرار الكود بحيث كل ما مسدج الايرور تظهر ويخليني اجرب الكود تاني احطه فinfinite loop
* بعد ما اعملها وانفذ الكود صح ومش عايزة الكود يتكرر ممكن اعمل break للكود

**EX:**

phone:  
while (true) {  
 try {  
 print('Enter your phone number:');  
 int a=int.*parse*(stdin.readLineSync()!);  
 break phone;  
 } catch (i) {  
 print('wrong value');  
 }  
}

while (true) {  
 try {  
 print('Enter your phone number:');  
 int a=int.*parse*(stdin.readLineSync()!);  
 break ;  
 } on FormatException {  
 print('wrong value');  
 }  
}

while (true) {  
 try {  
 print('Enter your phone number:');  
 int a=int.*parse*(stdin.readLineSync()!);  
 if(a == 5)  
 throw Exception();  
 break ;  
 } catch (i) {  
 print('wrong value');  
 }  
}

**9.List (1D list) :**

- نوع من انواع الdata type عبارة عن حاجة فيها مجموعة مالقيم (متشابهين فالdata type او لا)

بتتخزن فمتغير واحد

- اقواسها []

- اول قيمة بتاخد index(رقم او ترتيب) بيبدا من 0

- طول الlist بيبدا من 0 لx-1 وx عدد عناصر الlist

- بقدر اغير قيمة عنصر معين من عناصر الlist (mutable) او استخدمه من خلال اني بنده عليه من خلال الindex بتاعه

- اما يكون في list فيها قيم اقدر اعمل عمليات عليها باستخدام ميثودز زي :

9.1 first : بتجيب اول عنصر فالlist

9.2 last : بتجيب اخر عنصر فالlist

9.3 reversed : بتعكس ترتيب عناصر الlist

9.4 add() : بتاخد قيمة واحدة تضيفها فاخر الlist

9.5 addAll() : بتضيف قيمة او كذا قيمة فاخر الlist

9.6 insert() : زي الadd بس بتضيف عنصر واحد فمكان معين

9.7 insertAll() : زي addAll بس الفرق انها بتضيف عنصر او كذا عنصر فمكان معين

9.8 remove() : بتاخد قيمة عنصر موجود فالlist وتلغيه

9.9 removeAt() : بتاخد index عنصر موجود فالlist وتلغي العنصر

9.10 removeLast() : بتلغي اخر عنصر موجود فالlist

9.11 removeRange() : بتاخد range لindexes موجودة فالlist وبتلغي العناصر اللي فالindexes دي

9.12 replaceRange() :بتاخد range لindexes موجودة فالlist ومجموعة من عناصر تانية وتغير قيم العناصر اللي فيها بالعناصر التانية

9.13 isEmpty() : بتتاكد الlist فاضية ولا لا

9.14 isNotEmpty() : بتتاكد الlist مش فاضية ولا لا

9.15 indexOf() : بتجيب الindex بتاع عنصر معين

Note :

* الfor loop ممكن استخدمها فطباعة عناصر الlist بدل من جملة print() من خلال اني بعمل لوب عادية وببدا الvariable بتاعها من رقم 0 (اول index فالlist) واطبع الvariable
* لو انا معرفش طول الlist في حالة الطباعة بستخدم length()
* الfor statement بتستخدم مع الlists واشباهها وفكرتها اننا بنعمل variable بيلف علي كل عنصر من عناصر الlist وياخد قيمته ( عكس الfor loop العادية) وبستخدم القيمة دي بعدين
* الforEach بتستخدم مع الlist زي الfor statement فاني بعمل variable بيلف عالlist وبياخد قيمة كل عنصر

**EX:**

List x = [true, 1, "aliaa"];  
print(x.**isEmpty**);  
print(x.**isNotEmpty**);  
print(x.**first**);  
print(x.**last**);  
print(x.**reversed**);  
x.add(10);  
print(x);  
x.addAll([3, 'sd']);  
print(x);  
x.insert(1, 8);  
print(x);  
x.insertAll(2, [7, '\*']);  
print(x);  
x.remove('aliaa');  
print(x);  
x.removeAt(0);  
print(x);  
x.removeLast();  
print(x);  
x.removeRange(0, 3);  
print(x);  
x.replaceRange(0, 2, [1, 2, 3]);  
print(x);  
x[0] = 5;  
print(x);  
print(x.indexOf(5));  
for (int i = 0; i <= x.**length**; i++) {  
 print(i);  
}  
for (var i in x) {  
 print(i);  
}  
x.forEach((*b*) {  
 print(*b*);  
});

9.16 any : بتشوف اي قيمة بتحقق الشرط المعمول عالليست ونوعها bool

9.17 map : بتعدي علي قيم الليست وتعمل شرط او عمليات انا بكتبها فالbody بتاعها ومش بتتنفذ غير اما بستخدم المتغير اللي حطيت قيمة الميثود فيه ولو مثلا حطيت ليست جوا ليست وعايزة استخدم الماب فوانا بكتب الليست هكتب اندكس الليست اللي جوا اللي همر عليها

9.18 where : بكتب فيها شرط علي الlist اللي فيها القيم وبخزن الناتج فlist جديدة وبيكون كل القيم اللي بتحقق الشرط (ممكن اكتب الشرط والlist مباشر فالطباعة بدل ما اديها لمتغير)

9.19 firstWhere : بترجع اول قيمة بتحقق الشرط ولو مفيش قيمة بتحقق الشرط ممكن اضيفلها حاجة اسمها orElse بتنفذ شرط تاني

9.20 lastWhere : زي اللي فوق بالظبط بس بترجع اخر قيمة بتحقق الشرط

9.21 indexWhere : بترجع مكان اول قيمة بتحقق الشرط وممكن تاخد بداية السيرش فالليست من اني بحط index البداية

9.22 lastIndexWhere : زي اللي فوق بالظبط بس بترجع مكان اخر قيمة بتحقق الشرط

9.23 whereType : بترجع نوع الحاجة اللي انا عايزها ترجع مالقيم زي مثلا بترجع القيم الstring بس

9.24 retainWhere : بتعمل تعديل عالليست نفسها وهو انها بتخلي القيم اللي فالليست اللي بيحققوا الشرط بس عكس اللي فوق بيرجعوا القيم اللي بتحقق الشرط فليست جديدة (sublist) من الليست الاساسية من غير ما الاساسية تتاثر

9.25 removeWhere : زي اللي فوقها فالتعديل بس بتشيل القيم اللي بتحقق الشرط

9.26 singleWhere : بترجع قيمة واحدة بس بتحقق الشرط ولو في اكتر من قيمة بتحقق الشرط هترجع error وممكن عشان اتجنب الايرور بتاعها احطها فtry وcatch

**EX:**

List numbers = [1, 2, 3, 4, 5, 6];  
List set = [  
 [1, 2, 3, 4]  
];  
double sum = 0;  
double sum2 = 0;  
  
var a2 = numbers.map((*e*) {  
 sum2 += *e*;  
 return sum2;  
});  
var a3 = set[0].map((*e*) {  
 sum += *e*;  
 return sum;  
});  
  
var n1 = numbers.where((*element*) {  
 return *element* <= 4;  
});  
var n2 = numbers.lastWhere((*element*) => *element* >= 4, *orElse*: () => 0);  
var n3 = numbers.lastWhere((*element*) => *element* >= 4, *orElse*: () => 0);  
var n4 = numbers.indexWhere((*element*) => *element* <= 4, 3);  
var n5 = numbers.lastIndexWhere((*element*) => *element* <= 4, 3);  
var n6 = numbers.whereType<int>();  
var s = numbers.singleWhere((*element*) => *element* >= 4);  
numbers.any((*element*) => *element* >= 4);  
numbers.retainWhere((*element*) => *element* <= 4);  
numbers.removeWhere((*element*) => *element* <= 4);

**Iterable :**

زي data type اما متغير يكون واخدها معناه انه بيرجع كذا قيمة مش قيمة واحدة زي الlist والset والstring وغيره وممكن استخدمها كداتا تايب مباشر وبتكون اقواسها ()

ممكن اغير نوع متغير من list او غيره لiterable بس مينفعش العكس

**EX:**

Iterable set =[1,2,3,4];  
List a=[1,2,3,4];  
Iterable b=a;

**Spread operator : ( ... )**

- لو عندي مثلا 2lists وواحدة جزء مالتانية فبدل ما هكتب عناصر التانية كاملة ممكن احط الاولي فيها وقبلها ال"..." واكمل باقي العناصر فكأني بقوله يستخرج عناصر الليست الاولي ويحطها فالتانية

- في شكل تاني منه وهو دا (…?) ودا معناه لو الليست مثلا بnull سيبها متعملش فيها حاجة فدا بيمنع الايرور

Note : ممكن اكتب شرط قبل ال… مثلا يتاكد دي الليست المطلوبة ولا لا لو اه يستخرج عناصرها

**EX:**

List a=[1,2,3];  
var b=null;  
List c=[if(a.**length**==3)...a,4,5,...?b,6];  
print(c);

**10. 2D list :**

- زي الlist العادية بس الفرق انها 2d يعني بتتكون من صفوف واعمدة

- الindex بيتكون من رقمين رقم للصف ورقم للعمود زي الmatrix

- اقواسها [ [] ] (القوس الداخلي للصف الواحد والخارجي للlist كلها )

- نفس العمليات اللي بتتنفذ عال1d بتتنفذ عال2d

- اما باجي اطبعها باستخدام الfor loop بستخدم nested loop

- مش بينفع استخدم الlength معاها فالطباعة

- اما بستخدم الforEach او الfor statement بيعتبروا اقواس الصفوف انها عناصر الlist الاساسية ففالطباعة بيطبعوا صف بصف مش عنصر عنصر زي الnested loop

**EX:**

List x = [[true, 1, "aliaa"], [1, 2, 3]];  
print(x[1][1]);  
for (int i = 0; i <2; i++) {  
 for (int k = 0; k <= 2; k++) {  
 print(x[i][k]);  
 }  
}  
for(var f in x){print(f);}  
x.forEach((*element*) {print(*element*);});

**11. Sets :**

- نوع من الdata type شبه الlist من حيث انها بيتخزن قيم كتير فمتغير واحد

- اقواسها {}

- لو فيها عناصر متكررة بتحسهم كلهم كعنصر مع اول عنصر منهم يظهر

بعض عملياتها :

11.1 contains() : بتشوف في عنصر معين موجود ولا لا

11.2 elementAt() : بتشوف ايه العنصر الموجود عند index معين

11.3 intersection() : بتشوف التقاطع بين مجموعتين

11.4 union() : بتجيب الاتحاد بتاع مجموعتين

11.5 difference() : بتشوف فرق المجموعة الاولي عن المجموعة التانية

**EX:**

Set x = {1, 2, 'aliaa', 1, 2, 3};  
print(x.**length**);  
print(x);  
print(x.**isEmpty**);  
print(x.**isNotEmpty**);  
print(x.**first**);  
print(x.**last**);  
x.add(10);  
print(x);  
x.addAll([3, 'sd']);  
print(x);  
x.remove('aliaa');  
print(x);  
print(x.contains(0));  
print(x.elementAt(2));  
x.clear();  
print(x);  
  
Set y = {1, 2, 3, 4, 5};  
print(x.intersection(y));  
print(x.union(y));  
print(x.difference(y));  
print(y.difference(x));  
x.forEach((*i*) {  
 print(*i*);  
});

**12. Map :**

- نوع من الdata type شبه الlist والset فان بيتخزن قيم كتير فمتغير واحد

- شكل القيمة عبارة عن {key: value} وبياخدوا data type عادي

- بجيب الvalue من خلال الkey

- لو جيت استخدم الforEach فطباعة الmap هعمل متغيرين واحد للkey والتاني للvalue

بعض عملياتها :

12.1 keys() : هيجيبلي كل الkeys اللي فالmap

12.2 values() : هيجيب كل الvalues اللي فالmap

12.3 entries() : هيجيب الmap كلا بس فشكل MapEntries(key :value)

12.4 containsKey() : هيشوف في key معين موجود ولا لا

12.5 containsValue() : هيشوف في value معينة موجودة ولا لا

**EX:**

Map x ={1:'aliaa',2:4,'aliaa':5};  
print(x.**length**);  
print(x.**keys**);  
print(x.**values**);  
print(x.**entries**);  
print(x);  
print(x.**isEmpty**);  
print(x.**isNotEmpty**);  
x.addAll({3:'sd'});  
print(x);  
x.remove('aliaa');  
print(x);  
print(x.containsKey(0));  
print(x.containsValue(2));  
x.clear();  
print(x);

x.forEach((*i,n*) {  
 print(*i*); print(*i*);  
});

**13. Number methods :**

طرق بستخدمها عشان اعرف شوية معلومات عن الرقم زي هو فردي ولا زوجي وغيره :

9.1 isOdd : العدد فردي ولا لا

9.2 isEven : العدد زوجي ولا لا

9.3 isFinite : العدد محدود ولا لا

9.4 isInfinite : العدد ما لا نهاية ولا لا

9.5 isNaN : القيمة حاجة غير عدد ولا لا

9.6 isNegative : العدد سالب ولا لا

9.7 abs() : بجيب القيمة المطلقة للعدد

9.8 round() : بتقرب العدد لاقرب عدد صحيح ( مثلا 9.4 هيكون 9)

9.9 floor() : بتقرب لاصغر عدد صحيح (مثلا 9.4 هيكون 9)

9.10 ceil() : بتقرب لاكبر عدد صحيح (مثلا 9.4 هيكون 10)

9.11 truncate() : بتقرب لاقرب عدد للصفر (مثلا 9.7 هيكون 9)

9.12 remainder() : بتجيب باقي القسمة (مثلا 9.4 / 9 هيكون 0.4)

Note:

* من 1:6 بيتنفذوا علي اعداد int فقط
* الremainder بياخد قيمتين الاولي المقسوم والتانية المقسوم عليه

**EX:**

var x= 9;  
print(x.**isOdd**);  
print(x.**isEven**);  
print(x.**isFinite**);  
print(x.**isInfinite**);  
print(x.**isNaN**);  
print(x.**isNegative**);  
  
var z= 9.4;  
print(z.abs());  
print(z.round());  
print(z.floor());  
print(z.ceil());  
print(z.truncate());  
print(z.remainder(x));

**14. String methods:**

عمليات بعملها علي الstring :

10.1 isEmpty : النص فاضي ولا لا

10.2 isNotEmpty : النص مش فاضي ولا لا

10.3 toLowerCase() : بيخلي كل الحروف small

10.4 toUpperCase() : بيخلي كل الحروف capital

10.5 contains() : بتشوف في مقطع ما موجود فالنص ولا لا

10.6 replaceRange() : بتاخد range ليه بداية ونهاية وحروف او قيم string وبتبدل مجموعة الحروف اللي فالrange دا بالقيم اللي خدتها

10.7 replaceFirst() : بديها حرف واول ما يظهر فكلمة بيبدله بحرف او حروف تانية

10.8 replaceAll() : بديها حرف او قيمة من string عندي وحرف او قيمة تانية وكل ما الحرف يظهر هتشيله وتحط الحرف التاني

10.9 split() : بديها حرف معين وتعمل list وكل ما الحرف دا يظهر هتشيله وتفصل الكلمة لكلمتين عند مكان الحرف دا وهكذا لحد ما تفصل الكلمة كلها لكذا كلمة

10.10 join() :بتاخد حاجة string و لو عندي list فيها strings هتجمع كل عناصر الlist فكلمة واحدة وبين كل عنصر والتاني هتحط الحاجة الstring اللي خدتها فالاول

10.11 trim() : بتشيل اي مسافة موجودة فالكلمة

Note:

* النهاية بتاعة replaceRange() مش بتتغير يعني لو حاطة range مثلا من 0:2 فالحروف اللي هتتغير هتكون الاندكس بتاعها 0و1 فقط

**EX:**

var x= " aliaa";  
print(x.**isEmpty**);  
print(x.**isNotEmpty**);  
print(x.toLowerCase());  
print(x.toUpperCase());  
print(x.contains("a"));  
print(x.replaceRange(0, 7, "p"));  
print(x.replaceFirst('a', 'p'));  
print(x.replaceAll('a', 'p'));  
print(x.split('a'));  
print(['3','1','3'].join('a'));  
print(x.trim());

**15. Functions:**

- بتساعدني انظم الكود واقسمه علي اجزاء

- بدل ما اكرر الكود بستخدمها فاني بكتب كود معين وكل ما اعوز استخدمه اعمله call فقط

- بتتكون من جزئين :

**15.1** **definition:**

تعريف الدالة ودا بيتكون من جزئين:

1. header : بعرف فيه الdata type بتاع الfunction وهتاخد parameters ولا لا

2. body : بكتب فيه الجمل اللي هعوز الدالة دي تنفذها

**15.2** **calling :**

استدعاء الدالة فالmain عشان تتنفذ

**Syntax : main { function\_calling; }**

**Function\_definition;**

**طريقة عمل الfunction :**

بكتب الdefinition بره الmain وبعمل calling ليها فالmain فوقت التنفيذ البرنامج هينفذ من الmain هيلاقي الcalling بتاع الfunction فهيروح علي مكانها ويشوف فيها ايه وينفذه ويرجع تاني للmain بالناتج بتاع تنفيذ الfunction

**15.3** **انواع الfunctions من حيث الdata type :**

3.1 return value : بتاخد اي نوع data type وبترجعلي قيمة من نفس النوع دا

**Syntax: data\_type fun\_name (parameters/empty) => header**

**{ statements; => body**

**return value;}**

3.2 void : مش بترجع قيمة وبيكون نوعها void

**Syntax: void fun\_name (parameters/empty) => header**

**{ statements; } => body**

**15.4** **انواع الfunctions من حيث الparameters:**

parameters : variables بتتحط فاقواس الfunctions بتاخد قيمتها فالmain بعدين بس مقدرش استخدمهم كاسماء بره نطاق الfunction

4.1 بتاخد parameters :

**Syntax: void/data\_type fun\_name (data\_type par\_name)**

**{ statements; }**

4.2 مش بتاخد parameters :

**Syntax: void/data\_type fun\_name ()**

**{ statements; }**

Note:

* ممكن اكتب اسم الدالة واسم الparameters مباشر من غير نوعهم وساعتها بيكون نوعهم dynamic
* عشان اشوف قيمة الreturn لازم اطبعها
* ممكن اعمل variable وتكون الdata type بتاعته Funtion

**EX:**

void main(List<String> *arguments*) {  
 print(sum());  
 print(sub(4, 8));  
 div(6, 3);  
 mul();  
}  
  
int sum() {  
 return 5 + 6;  
}  
  
sub(int *x*, int *y*) {  
 return *x* - *y*;  
}  
  
void div(*x*, *y*) {  
 print(*x* / *y*);  
}  
  
mul() {  
 print(5 \* 6);  
}

**15.5** **انواع الParameters من حيث الrequired والoptional :**

في نوعين مالparameters نوع اجباري required ونوع اختياري optional :

15.5.1 positional parameters : اجباري(required) وهو اني بكتب اسم الparameter عادي ولازم اديله قيمته

15.5.2 optional positional parameter :اختياري(optional) وبحطه في [] ولازم احطه فاخر الparameters عشان ميحصلش ايرور عشان هو ثابت فمكانه (مثلا لو عندي 3 parameters الاول والتالت required والتاني optional وعايزة ادي قيمة للاول والتالت هيحسب اني بدي القيمة للاول والتاني وهيحصل ايرور عند التالت فلازم احط الrequired الاول بعد كدا الoptional

15.5.3 optional named parameters : اختياري (optional) وبحط الparameters في {} وبمرر قيمة الparameters بالاسماء (par\_name : value) ولازم احطها بعد الrequired

**15.6 انواع الParameters من حيث الvalue:**

15.6.1 default parameters : بيكون محطوط ليهم قيمة افتراضية فالتعريف بتاعهم بحيث لو اليوزر مدخلش قيمة فهيكون ليهم قيمة by default ولازم تكون محطوطة فاخر الparameters ايا كان نوعهم

15.6.2 no value : اليوزر بيديها او الparameters بتاخدها مالكود نفسه بعدين

**EX:**

void main(List<String> *arguments*) {  
 print(sub(4, 8));  
 mul(1, *z*: 5, *y*: 4);  
}  
  
  
  
sub(int *x*, int *z*, [var *y* = 10]) {  
 return *x* - *y* - *z*;  
}  
  
mul(*x*, {*z* = 10, *y*}) {  
 print(*y* \* *x* \* *z*);  
}

**15.7 Lambda expression :**

طريقة مختصرة لكتابة الfunction في حالة الbody كان سطر واحد

**Syntax: data\_type/empty fun\_name (parameters/empty) => statement**

**15.8 anonymous function :**

نوع من انواع الfunctions مش بيكون ليها اسم ولا data type وبتتكتب جوا الmain بس وممكن اكتبها فشكل lambda expression لو كان الbody سطر واحد

**Syntax: (parameters/empty) { statements;}**

Note :

* ممكن احط الlambda كقيمة لvariable بس لازم ساعتها احطها فشكل الanonymous

**EX:**

void mul() => print(5 \* 6);

var x= ()=>print(5);

**15.9 recursion :**

- نوع من انواع الfunctions بتنده علي نفسها تنفيذها بيبقي شبه تنفيذ الloop

- فكرتها اني بعملها call فالbody بتاعها

**EX:**

void main(List<String> *arguments*) {  
 print(sum(4));  
}  
  
int sum(*x*) {  
 if (*x* != 0)  
 return *x* + sum(*x* - 1);  
  
 return 0;  
}

**15.10 typedef :**

بستخدمها فتحديد نوع الfunction من حيث الparameters

مثلا عندي function اسمها fun1 محتاجة parameter من نوع function ويكون بياخد parameters هو كمان بس انا حطيت function تانية اسمها fun2 مش بتاخد parameters فكدا هيحصل عندي ايرور فالrun فبستخدم الtypedef فاني بعمل function (مثلاoperation) وبحدد فيها عدد الparameters فاما اجي اكتب الparameters فfun1 هكتب الparameter اللي هعمله function من نوع operation فكدا اتحدد اني هحط هنا function بعدد parameters معين

**Syntax: typedef fun\_name(parameters)**

**EX:**

void main(List<String> *arguments*) {  
 calc(4, 5, (add));  
}

typedef operation(*x*, *c*);  
calc(*t*, *y*, operation *g*) {  
 *g*(*t*, *y*);  
}

info() => print('aliaa');  
add(*a*, *v*) => print(*a* + *v*);  
sub(*a*, *v*) => print(*a* - *v*);  
mul(*a*, *v*) => print(*a* \* *v*);  
div(*a*, *v*) => print(*a* / *v*);

**15.11 Extension function :**

اما يكون عندي class لحاجة زي مثلا String class وانا عايزة استخدم فيه function بس هي مش فيه فبستخدم الextension function من خلال اني بضيف function من انشائي للclass واقدر استخدمها فاي وقت

**Syntax: extension extension\_name on class\_name {**

**Function\_name (parameters) {statements;}**

**}**

**EX:**

void main(List<String> *arguments*) {  
 var x = '2';  
 print(x.parseInt() + 5);  
}  
  
extension NumberParsing on String {  
 parseInt() {  
 return int.*parse*(this);  
 }  
}

**Note :**

* الthis دي بديل عن الobject او الvariable اللي هيستخدم الfunction دي (كبديل لاني اكتب parameters فالfunction)
* ممكن استخدم اسم الextension function فالcalling

**EX:**

void main(List<String> *arguments*) {  
 var x = '2';  
 print(NumberParsing(x).parseInt(x) + 5);  
}  
  
extension NumberParsing on String {  
 parseInt(*a*) {  
 return int.*parse*(*a*);  
 }  
}

**OOP**

البرمجة كائنية التوجه :

احنا فالطبيعي اما بنكتب كود بنكتبه ففايل واحد فبنحط كل حاجة فالفايل بجانب لو في جزء كود هستخدمه كذا مرة بكرره فكل مرة هستخدمه فالتعديل والكود هيبقي اسباجتي والteam work هيكون صعب ومن هنا جت فكرة الoop او الobject oriented programming وهي طريقة لتنظيم الكود عن طريق انها قسمت الكود لفايلات وكل فايل فيه بعض الحاجات زي الclasses فيها بعض الاكواد واحيانا بتكون ثابتة وممكن اعمل مالclasses دي object يورث كل حاجة فيها وحاجات غيرهم بالتالي الكود هيكون readable سهل ان حد يقراه ويعدل عليه وreusable سهل اني استخدم نفس الكود فحتة تانية مفيش فيه تكرار

**Class:** زي هيكل بانشئه بيكون فيه بعض الخصائص او الصفات (properties او attributes) وmethods وconstructors وغيرهم ويفضل اسم الclass يكون capital وبعمل منه objects بحيث ان الclass بيكون شيء عام والobjects بتكون خاصة وبدي قيم مختلفة لكل object

Syntax : class class\_name{}

**Object or instance:** حاجة بتورث كل حاجة مالclass وبضيف عليها حاجات تانية زي اني اديها قيمة فproperty معينة بتبقي عبارة عن الconstructor بتاع الclass (هيتشرح قدام)

Syntax: class\_name obj\_name= new class\_name();

Or class\_name obj\_name= class\_name(); (مش شرط اكتب كلمة new(

**Note**:

- ممكن اكتب data type بدل اسم الclass لان فالاصل كل الdata type هي class وبعمل منه object

Syntax: data\_type object\_name=();

**Anonymous object:** ممكن استخدم الobject من غير ما ادي قيمته لvariable يعني مثلا لو عندي class اسمه Car{} فانا ممكن استخدم الobject منه مباشر اللي هو Car() من غير ما ادي قيمته لvariable وبعدين استخدم الvariable علي انه الobject

**Attributes او properties:** خصائص او صفات بديها للclass وبديها قيمة بعدين اما بعمل object عبارة عن متغيرات بعرفها جوا الclass

Syntax: data\_type attribute\_name;

obj\_name.attribute\_name(value);

**Note:**

في طريقة مختصرة اما بستخدم الattributes لنفس الobject اسمها cascade بشيل فيها ال; من كل سطر ماعدا الاخير فمعني كدا ان السطر مخلصش وكل اللي هكتبه فسطر واحد

Syntax: class\_name object\_name=class\_name();

..attribute1=(value)..attribute2=(value)..method1=(value);

**Methods:** شبه الfunctions بنفس الsyntax بتاعها وهي افعال (actions) خاصة بالclass اما بعملها calling هتبدا تنفذ حاجة معينة فرقها عن الfunction ان الfunction بتبقي بره الclass والmain انما الmethod بتبقي جوا الclass

**Note** :

في طريقة مختصرة اما بستخدم الattributes والmethods لنفس الobject اسمها cascade بشيل فيها ال; من كل سطر ماعدا الاخير فمعني كدا ان السطر مخلصش وكل اللي هكتبه فسطر واحد

Syntax: class\_name object\_name=class\_name();

..attribute1=(value)..attribute2=(value)..method1=(value);

**EX:**

class Laptop {  
 String? **name**, **kind**, **color**, **productKey**, **processor**, **os**;  
 int? **id**;  
 double? **size**;  
  
 info() => print(  
 'name:$**name**\nkind:$**kind**\nprocessor:$**processor**\nsize:$**size**\nOS:$**os**\nproduct key:$**productKey**\nid:$**id**');  
}  
void main(List<String> *arguments*) {  
 Laptop lap1 = Laptop()  
 ..**name** = 'HP Elitebook 745 g3'  
 ..**kind** = 'HP'  
 ..**os** = 'Windows'  
 ..**id** = 12345  
 ..**processor** = 'AMD A10-8600'  
 ..**size** = 14  
 ..**productKey** = '00000-00000-00000-00000'  
 ..**color** = 'Silver'  
 ..info();  
}

**Constructors**:

- نوع خاص من الmethods ليه نفس خواص الfunctions بالظبط بس مختلف فانه اسمها لازم يكون بنفس اسم الclass مهمته اني بعمل initialization للvariables بتاعة الclass

Syntax:className(parameters){statements;}

**EX:**

class Laptop {  
 static String? *name*, *kind*, *color*, *productKey*, *processor*, *os*;  
 int? **id**;  
 double? **size**;  
  
 Laptop(  
 {String? *c\_name*, *c\_kind*, *c\_color*, *c\_productKey*,  
 *c\_processor*, *c\_os*,  
 int? *c\_id*,  
 double? *c\_size*}) {  
 *name* = *c\_name*;  
 *kind* = *c\_kind*;  
 *color* = *c\_color*;  
 *productKey* = *c\_productKey*;  
 *processor* = *c\_processor*;  
 *os* = *c\_os*;  
 **id** = *c\_id*;  
 **size** = *c\_size*;  
 print(  
 'name:$*name*\nkind:$*kind*\nprocessor:$*processor*\nsize:$**size**\nOS:$*os*\nproduct key:$*productKey*\nid:$**id**');  
 }  
}  
  
void main(List<String> *arguments*) {  
 Laptop lap1 = Laptop(  
 *c\_name*: 'HP Elitebook 745 g3', *c\_processor*: 'AMD A10-8600', *c\_size*: 14);  
}

}

وممكن اكتبه من غير ما اساويه بالattributes فالحالة دي هيكون الattributes ملهمش لازمة بس الطريقة دي مش بتسخدمها كتير عشان الattributes نفسها ممكن احتاجهم بعدين فالكود فمش هعرف اوصلهم لو كانوا parameters بس فالconstructor فبعرفهم برده

**EX:**

class Laptop {  
 Laptop(  
 {String? *name*,  
 *kind*,  
 *color*,  
 *productKey*,  
 *processor*,  
 *os*,  
 int? *id*,  
 double? *size*}) {  
 print(  
 'name:$*name*\nkind:$*kind*\nprocessor:$*processor*\nsize:$*size*\nOS:$*os*\nproduct key:$*productKey*\nid:$*id*');  
 }  
}  
  
void main(List<String> *arguments*) {  
 Laptop lap1 = Laptop(*name*:'HP Elitebook 745 g3',*processor*:'AMD A10-8600',*size*: 14 );  
}

وليه نوعين :

**default constructor:** بيتنفذ اول ما بعمل الobject زي مثلا انه يطبعلي جملة علي طول من غير ما ياخد حاجة

**named constructor(constructor with parameters)** : بياخد parameters بعملها calling بعدين وبديها قيمتها

**syntax:**

**class\_name.any\_name(parameters){statements;}**

**static:**

بكتبها قبل الattributes والmethods بحيث اقدر اوصلهم فالmain من غير ماعمل object مالclass بتاعهم فكدا كأني بقول ان الmethods والattributes دي تبع الclass بتاعهم يعني مينفعش نوصلهم الا عن طريق الclass مش object

Syntax: static attribute\_name;

static method\_name(){}

class\_name.attribute/method\_name();

**EX:**

class Laptop {  
 static String? *name*, *kind*, *color*, *productKey*, *processor*, *os*;  
 static int? *id*;  
 double? **size**;  
}  
  
void main(List<String> *arguments*) {  
 Laptop.*name*='HP Elitebook 745 g3';  
 Laptop.*processor* = 'AMD A10-8600';  
 Laptop.*productKey*='00000-00000-00000-00000';  
 Laptop.*id*=12345;  
 Laptop lap1=Laptop();  
 lap1.**size**=14;  
}

**this:**

كأنها رمز للobject بستخدمها عشان اربط قيمة الattribute بقيمة الparameter بتاع الmethod او الconstructor

Syntax: this.attribute\_name=constructor/method\_name;

Note:

لو خليت الattribute نوعها static مش هعرف استخدم this معاها لاني كدا هربطها بالobject وstatic بتخلي الوصول للattribute من خلال الclass يعني مفيش object فالموضوع

**EX:**

class Laptop {  
 String? **name**, **kind**;  
 static int? *id*;  
  
 Laptop({String? *c\_name*, *c\_kind*, int? *c\_id*}) {  
 this.**name** = *c\_name*;  
 this.**kind** = *c\_kind*;  
 *id* = *c\_id*;  
  
 print('name:$**name**\nkind:$**kind**');  
 }  
}  
  
void main(List<String> *arguments*) {  
 Laptop lap1 = Laptop(*c\_name*: 'HP Elitebook 745 g3');  
}

**Encapsulation:**

- المبدا الاول فالoop

- تغليف بمعني ان الdata بتاعتي (زي الattributes ) عايزة اوفرلها نوع مالحماية لكن دا مش حماية مطلقة انا بحاول احافظ عليها لكن ممش بحميها كليا فكأني بحطها جوا layer او cover عشان موصلش ليها مباشر عن طريق الaccess modifier اني بحدد هل الattribute دي اقدر اوصلها مباشر ولا لا

**Access modifier:**

- عبارة عن كلمات بحدد بيها اقدر اوصل للattributes مباشر ولا لا وهي public , private, protected

- الكلمات دي مش موجودة فالdart لكن في بديل ليها وهو ال\_ بحطه قبل اسم الattribute وبيعبر عن الprivate ولو مش موجود يبقي الattribute عبارة عن public

- لو الattributes كانت private بقدر اديها value من خلال طريقتين :

**1.** ي اما هعمل class فنفس الdart file اللي انا فيه وساعتها هقدر اوصل للprivate attribute من خلال اني استخدمها مباشر زي ما بوصل للattribute العادية

**2.** ي هعمل الclass فdart file تاني وساعتها هستخدم الget & set methods عشان اقدر احط value ليها

**set & get :**

**set** : method بستخدمها فاني بحط قيمة للattribute من خلال الparameters (بيكون نوعها void) وبستخدمها اكتر مع الprivate attributes

Syntax: set method\_name(parameter){

this.attribute\_name=parameter;}

**get** : method بستدعي قيمة الattribute اللي حطيتها فالset

Syntax: get method\_name{

return attribute\_name;}

**EX:**

class Laptop {  
 String? **\_name**;  
  
 set **name1**(*c\_name*) {  
 this.**\_name** = *c\_name*;  
 }  
  
 get **print\_name** {  
 return **\_name**;  
 }  
}  
  
void main(List<String> *arguments*) {  
 Laptop lap1 = Laptop();  
 lap1.**print\_name**('HP Elitebook 745 g3');  
}

**Inheritance:**

- المبدا التاني فالoop

- مبدا الوراثة بطبقه اما بكون عندي class فيه اكواد وعايزة استخدمها فclasses تانية فبدل ما اكرر الكود بورث من الclass الاول كل اكواده للclasses التانية فكدا الclasses التانية شايفة اكواد الclass الاول وقادرة تستخدمها

- بقول علي الclass الاول اللي وارث منه super class والclass التاني اللي ورث من الاول sub class

- لو عندي اكتر من 2 classes وارثين من بعض فبقول عالclass اللي وارث من class تاني اسم sub class والclass اللي وارث منه مباشر direct super class والclass اللي وارث منه بصورة غير مباشرة indirect super class

- مينفعش الclass الواحد يورث من 2 classes فنفس الوقت(الجملة) لازم علي مراحل

**Syntax: class class2\_name extends class1\_name{}**

**EX:**

class Laptop {  
 *// indirect super class* String? **name**, **kind**, **color**, **productKey**, **processor**, **os**;  
 int? **id**;  
 double? **size**;  
}  
  
class HP extends Laptop {  
 *// direct super class*}  
  
class HP\_Elitebook extends HP {  
 *//sub class*}  
  
void main(List<String> *arguments*) {  
 Laptop lap1 = HP\_Elitebook();  
 lap1.**size**=14;  
}

**override:**

- ميثود بتخليني اعدل علي قيمة method او function من غير ما اغير فالdefinition بتاعها (مثلا عندي ميثود اسمها print\_info() فclass Laptop{} فاقدر اعدل علي قيمة الbody فclass تاني غير Laptop بس لازم الclass التاني يكون وارث من الLaptop )

Syntax: class class1\_name{

method1(){statements;}

}

class class2\_name{

method1(){statements;

}نفس اسم الميثود اللي فوق بس الجمل بتاعتها متغيرة

}

**EX:**

class Laptop {  
 String? **name**;  
   
 print\_info() {  
 print('name=$**name**');  
 }  
}  
  
class HP extends Laptop {  
 *@override* print\_info() {  
 print('aliaa');  
 }  
}  
void main(List<String> *arguments*) {  
 HP lap1 = HP();  
 lap1.print\_info();  
 var lap2 = Laptop();  
 lap2.print\_info();  
}

**super:**

ممكم استخدمها قبل الmethods او الattributes اللي وارثاها من class فclass تاني كتحديد ان الحاجات دي تبع الclass اللي وارثة منه مش الclass اللي انا فيه

**EX:**

class Laptop {  
 String? **name**;  
  
 print\_info(*name*) {  
 print('name=$*name*');  
 }  
}  
  
class HP extends Laptop {  
 play(*c\_name*) {  
 super.print\_info(*c\_name*);  
 }  
}  
  
void main(List<String> *arguments*) {  
 var lap1 = HP();  
 lap1.play('hp elitebook');  
}

**super constructor:**

لو عندي 2 classes : Laptop,HP والHP وارث مالLaptop والاول فيه constructor ساعتها هيحصل ايرور عشان مينفعش اورث constructor فبلجا لطريقة الsuper اني بورث الconstructor لconstructor الHP بحيث ان كلمة super بتعبر عن Laptop constructor

Syntax: class class1\_name{

class1\_name(){}

}

class class2\_name{

class2\_name():super(){}

}

**EX:**

class Laptop {  
 Laptop(String? *name*) {  
 print(*name*);  
 }  
}  
  
class HP extends Laptop {  
 HP(*n*) : super(*n*) {}  
}  
  
void main(List<String> *arguments*) {  
 var lap1 = HP('HP Elitebook');  
}

**Abstraction:**

- المبدا التالت فالoop

- مبدا بيعتمد علي انشاء class مقدرش اعمل منه objects من خلال اني بكتب قبل الclass كلمة abstract وعشان اعمل منه object بورثه لclass تاني وبعمل object مالclass التاني

- بستخدم المبدا دا لو مثلا عايزة اعمل attributes وmethods هستخدمها بعدين (زي توريثها لclasses تانية) بس مش محتاجة اعمل object مالclass دا

- اما بكتب method فالabstract class مش بكتب body (بس عادي لو كتبته مش هيحصل error)

- لازم اعملoverride فالclass التاني لكل method عملتها (بس لو كتبت body مش لازم )(بس الconstructor مش شرط )

**Syntax:**

**abstract class class1\_name{**

**attribute1;**

**method1();}**

**class class2\_name{**

**@override**

**method1(){statements;};**

**}**

**EX:**

abstract class Laptop {  
 Laptop();  
 String? **name**;  
 print\_info();  
}  
  
class HP extends Laptop {  
 *@override* print\_info() {  
 print('HP Elitebook');  
 }  
}  
  
void main(List<String> *arguments*) {  
 var lap1 = HP();  
 lap1.print\_info();  
}

**Interface**:

- بتخليني اورث من اكتر من class فنفس الجملة

- لازم اعمل override لكل الattributes والmethods الموجودة فكل الclasses اللي وارثة منها

- الmethod بكتبها من غير body (لان لو كتبت body هيحصل error بس فعليا الinterface مش موجود فdart فبنتعامل معاه علي انه abstract وعشان اعرف الclass الل هيورث منه انه interface بغير كلمة الوراثة بدل extend تكون implement)

- الconstructor مش لازم يتعمله override

**Syntax:**

**abstract class class1\_name{**

**attribute1;**

**method1();}**

**abstract class class2\_name{**

**method2();**

**}**

**class class3\_name{**

**@override**

**attribute1;**

**@override**

**method1(){statements;};**

**@override**

**method2(){statements;};**

**}**

**EX:**

abstract class Laptop {  
 Laptop();  
 String? **name**;  
 print\_info(*name*);  
}  
  
abstract class HP {  
 print\_info1();  
}  
  
class HP\_Elitebook implements Laptop, HP {  
 *@override* String? **name**;  
  
 *@override* print\_info(*c\_name*) {  
 **name** = *c\_name*;  
 print('name=$**name**');  
 }  
  
 *@override* print\_info1() {  
 print('Silver');  
 }  
}  
  
void main(List<String> *arguments*) {  
 var lap1 = HP\_Elitebook();  
 lap1.print\_info('HP Elitebook');  
 lap1.print\_info1();  
}

**Mixin :**

- نوع من انواع الabstract class فبالتالي مش بقدر اعمل منه object ومش بكتب body الmethod(بس لو كتبتها مش هيحصل error)

- بقدر اخلي class يورث منه من خلال كلمة with وبقدر اورث من كذا class فنفس الجملة

- مش لازم اعمل override لكل الmethods يعني ممكن اخد حاجات معينة مالmixin وانا بورث منه مش لازم كلها

- لو هيورث من abstract class بستخدم on

**EX:**

abstract class Laptop {  
 Laptop();  
 String? **name**;  
 print\_info1();  
}  
  
mixin HP on Laptop {  
 *@override* print\_info2() {  
 print('HP Elitebook');  
 }  
}  
  
mixin Dell on Laptop {  
 *@override* print\_info3() {  
 print('Dell');  
 }  
}  
  
class Elitebook extends Laptop with HP, Dell {  
 *@override* print\_info1() {  
 return print('hp Elitebook g745');  
 }  
 *@override* print\_info2() {  
 return print('Elitebook g730');  
 }  
}  
  
void main(List<String> *arguments*) {  
 var lap1 = Elitebook();  
 lap1.print\_info1();  
 lap1.print\_info2();  
}

**Final VS Const :**

- نوع من انواع الdata type

- المشترك بينهم :

1. ممكن اكتب نوع data type معاهم زي String مثلا وممكن لا

2. مينفعش اغير قيمة الvariable بعد ما عملتله assign (اديته قيمة)

3. لازم الvariable ياخد قيمة اول ما اعرفه (اعمله initialization)

- الفرق بينهم:

Final:

1.معناه اني بثبت الinitialization بتاع الvariable بس يعني بثبت نوع الdata type بتاعته واسمه لكن اقدر اعدل علي قيمته مستقبلا عادي (زي الlist والعمليات عليها زي الadd وغيره اقدر اعملها)

2. عادي الvariable ياخد قيمته فالcompile time او الrun time (وقت كتابة الكود او وقت الrun واني اشوف نتيجة الكود) زي مثلا اني استخدم الDateTime.now() فكدا الvariable هياخد قيمته وقت الrun لانه هياخد تاريخ اليوم الحالي لكن انا مديتهوش القيمة بايدي ثابتة

3. لو عملت class اقدر اكتبها مباشر من غير كلام قبلها

4. اقدر مديش للvariable بتاعه قيمة بس بشرط اني اديله قيمة فmethod سواء كانت method عادية او constructor

Const:

1. معناه اني بثبت الinitialization بتاع الvariable والقيمة بتاعته يعني مينفعش اعدل عالقيمة بتاعته فالمستقبل نهائي(زي الlist والعمليات عليها زي الadd وغيره مقدرش اعملها)

2. الvariable بياخد قيمته فالcompile time فقط يعني مثال الdateTime معاه هيعمل error

3. لو عملت class لازم اكتب static قبل const

**Enum:**

- class شبه الconst والfinal بس لمجموعة من القيم فهو يعتبر class من الثوابت

**EX:**

void main(List<String> *arguments*) {}  
  
enum Laptop { **HP**, **Dell**, **Lenovo** }

**Generic:**

لو مثلا عندي variables في class مش عايزة احدد الdata type بتاعتهم او methods مش عايزة احدد نوع الdata type بتاعة الparameters بتاعهم فبعمل حاجة اسمها generic اني بكتب اي كلمة او حرف فال(diamond brackets) <> وبعدين و ادي data type للmethods والvariables بنفس نوع الgeneric ووقت ما اعمل calling للmethods او object مالclass بدي قيمة للvariables والparameters والcompiler تلقائي هيتعرف علي نوعهم من قيمتهم وممكن اعمل calling كذا مرة وكل مرة بقيمة مختلفة

Note:

لو مثلا كان عندي class وكتبت ليه نوع generic او اكتر وحددت نوع الgeneric بتاعه وانا بعمل منه object لازم اكتب نوع الgeneric كلهم او مكتش خالص

**EX:**

void main(List<String> *arguments*) {  
 var l1 = Laptop();  
 l1.**id** = 123;  
 var l2 = Laptop();  
 l2.**id** = '123';  
 var l3 = Laptop();  
 l3.**id** = 123.1;  
 print1(1);  
 print1('1');  
}  
  
class Laptop<lab> {  
 lab? **id**;  
}  
  
print1<pr>(pr *version*) {  
 print(*version*);  
}

**Asynchronous programming**

**Future:**

- احيانا بعوز انفذ اكواد معينة قبل او بعد اكواد تانية فدا بسميه الasynchronous او عدم التزامن فمن ضمن طرق تنفيذ الموضوع دا الfuture معناها كود هيتنفذ فالمستقبل بعد باقي الاكواد

- الfuture عبارة عن abstract class فعشان استخدمه بعمل object منه بطريقة غير مباشرة زي اني بعمل object مالconstructor بتاعه

- من انواع الconstructors بتاعة الfuture class :

**1. Future() :**

-ودا بياخد function

- لو ال function نوعها anonymous او نوعها void فالfuture هتتنفذ عادي

- لو الfunction بترجع قيمة ساعتها الfuture مش هتتنفذ لانها هتكون غير مكتملة فممكن اخليها تكتمل بطريقتين :

.then() :يعني الfuture هترجع قيمة فانا هستخدم القيمة دي فانها تكتمل وانفذ كود الfuture وبتاخد function وبكتب الاكواد اللي الfuture هينفذها جواها و ليها parameter بيشير للقيمة اللي هترجع من الfunction الاساسية بتاعة الfuture

.catchError() : لو الfuture هترجع error فانا ممكن استخدم قيمته فاني انفذ كود الfuture وزي اللي فوق بس الparameter هنا هيعبر عن الerror مش القيمة اللي هترجع من الfunction الاساسية بتاعة الfuture

**2. Future.value() :**

بتاخد قيمة محددة سواء variable او function

**3. Future.delayed() :**

باخر الfuture لمدة معينة انا بحددها وبتاخد 2 parameters مدة زمنية وfunction وممكن مكتبش الfunction

**EX:**

void main(List<String> *arguments*) {  
 var x =2;  
 var f1=Future(print1);  
 var f2=Future(print2).then((*value*) => print(print2()));  
 var f3=Future.value(print3());  
 var f4 =Future.delayed(Duration(*seconds*: 2),(){print(4);});  
 print(0);  
}  
print1()=>print(1);  
print2()=>2;  
print3()=>print(3);

**Stream:**

شبه الfuture بس هنا بتعامل مع مجموعة من القيم مش قيمة واحدة

من انواع الconstructors بتوعه :

**1. Stream.peroidic() :**

- كل مدة مالزمن هينفذلي حاجة او هيديني قيمة معينة حسب الاوامر اللي هديهاله

- بياخد 2 parameters مدة وfunction والfunction لو خدت parameter ومخدش قيمة تلقائي قيمته هتبدا من 0 وتزيد

- عشان اقدر استخدم قيمة اللي هترجع مالstream بستخدم الlisten :

- معناها ان كل قيمة هتجيلي مال stream استخدمها فالاوامر اللي هكتبها فالlisten

- بتاخد parameters:

* onData() :ودا required وهو اللي بيشتغل عالقيم اللي راجعة مالstream
* onError() : ودا named ودا في حالة حدوث ايرور
* onDone() : ودا named ودا اما الstream يكتمل
* cancelOnError() : ودا named ودا لو حصل ايرور هل هكمل الstream ولا هوقفه (هوقف اني اخد قيم منه لكن هو مكمل عادي)

**EX:**

void main(List<String> *arguments*) {  
 var x = Stream.periodic(Duration(*seconds*: 1), (*a*) {  
 return *a*;  
 }).listen((*event*) {  
 print(*event*);  
 }, *onDone*: () => print('done'), *onError*: (*error*) => print(*error*));  
}

Note :

ممكن انفذ where function علي الstream كنوع من انواع الfilteration

**Async , Await :**

- بتشتغل مع ال الfuture والsteam

- فايدتها اني بقلل اكواد الstream وال future والfunctions بتاعتهم من خلال اني بحول المكان اللي بكتب فيه الكود لasync وبكتب await قبل الكود اللي انا محتاجاه يتنفذ قبل باقي الاكواد يعني بخلي مثلا كود الfuture يتنفذ قبل ما باقي الاكواد تتنفذ

- المفروض الكود فالطبيعي بيكون نوعه future او stream فاما بستخدمهم هما بيحولولي الكود لكود طبيعي (مثلا لو انا عندي future<int> اما هستخدمهم هيحولولي الكود لint فوقت التنفيذ هيتعامل علي انه int وفالstream هيحوله لiterable اللي هو list مثلا) لكن اقدر استخدم الasync والawait من غير وجود الfuture والstream

**EX:**

void main(List<String> *arguments*) async {  
 var f4 = await Future.delayed(Duration(*seconds*: 2), () => 4);  
 print(f4);  
   
 var x = Stream.periodic(Duration(*seconds*: 1), (*a*) => *a*);  
 await for (int i in x) {  
 print(i);  
 }  
}

**API**

- اختصار لapplication program interface

- وظيفته الاساسية ان في داتا عندي عالنت فداتا بيز مثلا وانا عايزة استفيد منها فالابلكيشن بتاعي فهو بيعمل ربط بين الداتا بيز والابلكيشن

- مكتوب بالjson وفي مواقع بتوفر fake api زي [JSONPlaceholder - Free Fake REST API (typicode.com)](https://jsonplaceholder.typicode.com/?msclkid=7566eb76c5ac11ec82529f1e1bc2f551) عشان لو حد عايز يتدرب عليها موجودة فجزء ال resources و فيه الاوامر (الميثودز) بتاعة الhttp اللي نقدر نتعامل بيها مع الapi فجزء الroots

- الداتا فيه بتبقي علي شكل map اللي هو {key :value} وبتكون كلها string فبعملها decode للbody بتاع الداتا فالكود بتاعي عشان ارجعها لاصلها

- المفروض عشان اقدر اتعامل معاه انزل ال http package وببدا اعمل http request بكذا طريقة هتظهر فالmethods

- في methods فالhttp عشان اقدر اتعامل مع الapi زي:

**1.get() :**

- لو انا عايزة اجيب الداتا الللي فالapi عندي فالبروجكت واقدر استخدمها

- بتاخد object نوعه Uri وبحط فيها لينك الapi اللي عايزة اخد منه الداتا بس اللينك هيكون string فهعمله parsing

**Syntax: get(Uri.parse(‘Link’ ));**

Note :

- احيانا الميثودز اللي فالhttp package بتتشابه مع ميثودز فباكدجات تانية فعشان اخصص الموضوع بدي اسم للpackage وبستخدم الميثودز اللي فيها من خلال الاسم الجديد فيما يعرف بالaliasing او الcascade

- المفروض اما بتعامل مع الapi بيكون حاجة هتاخد وقت فالمستقبل يعني يعتبر future فاقدر اني استخدم معاه الasync والawait لو الكود معتمد عالداتا اللي هترجع منه

- ممكن عشان اخصص التعامل اكتر بشوف حالة الكود او فيما يعرف بالstatus code بمعني لو الداتا تمام وهترجعلي قيمة يبقي من نوع 200 لو الموقع فيه مشكلة او ايرور هيرجعلي 400 او 404 حسب نوع المشكلة وبناءا عالحالة بكتب الaction اللي هيحصل الكود

**EX:**

import 'package:http/http.dart' as http;  
import 'dart:convert';  
  
void main(List<String> *arguments*) async {  
 var data = await fetchData();  
 for (var i = 0; i < 100; i++) {  
 print('id=${data[i]['id']}');  
 print('title=${data[i]['title']}');  
 print('body=${data[i]['body']}');  
 print('----------------------------------------');  
 }  
}  
  
fetchData() async {  
 var response =  
 await http.get(Uri.*parse*('https://jsonplaceholder.typicode.com/posts'));  
 if (response.**statusCode** == 200) {  
 var object = json.decode(response.**body**);  
 return object;  
 } else {  
 throw Exception('Error');  
 }  
}

**Import**

اما باجي اكتب كود مينفعش اكتبه كله فنفس الfile عشان ميحصلش لخبطة وعدم تنظيم فببدا اقسم الproject عندي لمجموعة مالfiles والdirectories(folders) بحيث العملية تبقي منظمة اكتر وبستخدم اللي جواهم من خلال اني بعمل حاجة اسمها import وهي بتخليني استخدم اللي جوا الfiles والdirectories من غير ماكتب اللي جواهم فنفس الفايل اللي انا شغالة فيه

او لو انا هكتب كود حاجة والحاجة دي بتحتاج package عشان تنفذها فبعمل import للpackage عشان اقدر انفذ الكود (زي اني استقبل قيمة ماليوزر لازم اعمل import للio package )

**EX:**

class Laptop{  
 int? **id**;  
}  
var r;

import 'dart:io';  
import 'study/aliaa.dart';  
void main(List<String> *arguments*) {  
 var a= stdin.readLineSync()!;  
 var lap=Laptop();  
 lap.**id**=123;  
 r=3;  
}

**Dart packages**

احيانا بحتاج اني اعمل حاجات فالكود بس مش بتكون موجودة فالدارت فبضطر اني انزل حاجات فالكود بتاعي تعملها الحاجات دي بقول عليها packages الدارت بتكون موفراها للتنزيل

- فبروح لموقع [Dart packages (pub.dev)](https://pub.dev/)

- بسيرش فيه علي اسم الpackage اللي محتاجاها وبشوف هي هتشتغل فنفس المكان اللي هشغل عيله البروجكت ولا لا يعني مثلا لو هجرب الكود عالويب فبشوف هي بتشتغل مع الويب ولا لا

- بلاقي معلومات عن الباكدج زي استخدامها وهي ايه وغيره ومن ضمنها التنزيل بتاع الباكدج

- بشوف فجزء الinstall او التنزيل بنزلها ازاي لو شغالة دارت ليها امر ولو فلاتر ليها امر او اني اضيفها ففايل الpubspec.yaml من خلال الdependencies واعمل pub get عشان يتعرف عالتغييرات اللي حصلت

- بعد ما بنزلها بعملها import فالمكان اللي عايزة استخدم الpackage فيها وبشوف ايه الاكواد اللي محتاجاها منها وابدأ استخدمها

Note :

-احيانا بيكون في extensions بتعمل هي الحاجات دي تلقائي بس بتبقي حسب الide اللي شغالة عليه

- كل package ليها version وكل مرة ممكن تتغير فلازم اكون عارفة الفيرجن الحالية ايه عشان ميحصلش ايرور فالكود

- ممكن وانا بنزل الpackage احط اسمها بس فالpubspec.yaml واعمل pub get وهو هيجيب اخر version ليها :

provider:

**File system**

- عبارة عن class abstract فبالتالي هعمل منه object مالconstructors بتوعه

- بستخدمه فالتعامل مع الfiles

- عشان اقدر اتعامل معاه لازم اعمل import لdart.io وهي اختصار لinput/output علي اعتبار اني هدخله ويرجعلي files

- من اهم الconstructors اللي هتعامل معاها :

**1. File(String path) :**

بتاخد الفايل اللي هتعامل معاه

- معظم الmethods اللي موجودة فيه نوعها future وليها نوعين :

نوع عادي :ودا بستخدمه لو انا هستخدم async والawait او لو هتعامل مع الfuture methods زي then

نوع فيه sync:ودا كبديل للasync والawait والfuture methods (مثال existSync())

- من الmethods اللي موجودة فيه :

**1. exist() :**

بتشوف الfile موجود ولا لا

**2. delete() :**

بتلغي الfile لو موجود

**3. create() :**

بتعمل file جديد

**4. writeAsString() :**

بتكتب جوا الfile

**5. readAsString() :**

بتقرا اللي جوا الفايل

**6. copy(String newPath) :**

بتنسخ الfile وبتاخد اسم او مسار النسخة

لو مديتهاش مسار مختلف هتنسخ الfile فنفس الfolder او المكان اللي الفايل الاساسي فيه

**EX:**

import 'dart:io';  
  
void main(List<String> *arguments*) async {  
 var file = File('aliaa.txt');  
 print(await file.exists());  
 file.createSync();  
 file.copySync('asaad.txt');  
 file.writeAsStringSync('test');  
 print(file.readAsStringSync());  
}

**Null safety**

- في نسخ الdart القديمة اما كنت اجي اعرف variable مكنش يقبل اني اديله قيمة بnull (كأني بقوله المتغير مفيش فيه قيمه ) بس بعد تحديث flutter بقي في مفهوم الnull safety وهو اني اقدر ادي للvariable بتاعي قيمة null

- بس فالمقابل لازم احط (?) بعد ال variable data type ودي معناها اني بقوله ان الvariable دا ممكن قيمته تكون null او نوع الdata type اللي معموله initialization بيها

- لو انا عرفت الvariable واديته قيمة مش بnull ساعتها مش لازم احط ال?

- لو انا محطتش ? جمب الdata type وفي احتمال ان قيمته تكون بnull هيعمل error

- لو عرفت variable جوا method من غير ما ياخد قيمة اقدر محطش ال? عادي لانه وقتها اللي ظاهر انه مش هياخد قيمة سواء بnull او بغيرها فبالتالي مش هيحصل error

- انما لو فclass فلازم احط ال?

- لو فرضا استخدم condition زي if علي قيمة variable مكنتش اديته قيمة لسه ومحطتش else هيعمل error لان كدا في احتمال ان لو الif متنفذتش فهو كدا هيكون من غير قيمة يعني null فلازم احط ?

- لو انا عرفت variable وانا متاكدة ان قيمته اللي هياخدها مش هتكون بnull بحط ! جمب قيمته بس ساعتها ممكن يحصل error لان ممكن قيمته تكون بnull فالكود نفسه من غير ماعرف او اخد بالي

- في مفهوم اسمه soundness null safety دا معناه ان اللغة نفسها بتتاكد ان كل الvariables اللي فالكود مش هتاخد قيمة null وهي مش متاح ليها دا مش بتسيب الموضوع لوقت الrunning او بعد التشغيل ويحصل run time error ان الكود ميكلمش فناتج الrun بسبب error حصل فالكود

**EX:**

import 'dart:io';  
void main(List<String> *arguments*) async {  
 String a='';  
 String b;  
 String? c=stdin.readLineSync();  
 String d=stdin.readLineSync()!;  
}

**Required & late**

**1. required:**

- بستخدمها لو في حاجة optional وانا عايزة اخليها اجباري انها تتنفذ ( مثلا عندي named parameter وانا عايزة اخليه اجباري انه يتكتب بدل طريقة الpositional او بدل ما احط ? بعد الdata type بتاعته)

- بيفيدني احيانا فتجنب null error بعد تحديث الdart وانها مش بتقبل وجود variable قيمته بnull غير فظروف معينة

**2. late:**

لو انا عندي مثلا variable فclass وهستخدمه بس لسه محطتش ليه قيمة فهيحصل error عشان الdart هتشوفه null variable وبستخدمه فحاجة فبستخدم كلمة late كأني بقولها اني هحطله قيمة بعدين ومتعمليش error (دا بديل لل? بعد الdata type)

**EX:**

void main(List<String> *arguments*) async {  
 var hp = Laptob(*name*: 'hp', *id*: 12345);  
}  
  
class Laptob {  
 late int **id**;  
  
 Laptob({required String *name*, required this.**id**}) {  
 print(*name*);  
 }  
}

**Notes :**

DateTime.now().year : بيجيبلي التاريخ الحالي بالسنة وممكن اغيره عادي

* لو عندي function لازم تاخد function parameter وانا مش عايزة اديها الparameter دلوقتي بحط\_ مكان الparameter الfunction و\_ مكان الreturn وبعدين بقدر اديهولها عادي

**Syntax : (\_) {return \_;} or (\_) =>\_**