

## Types

→ boolean ↔

```

bool b;
b = true; // strongly typed
print(b); // true
// optional
var b;
print(b); // null
b = false;
print(b); // false
// untyped type
var b = false;
print(b); // false

```

→ numeric ↔

```

int x = 10; // x = int
var y = 123; // y = int
num z = 1; // z = int
var t = int. parse("5"); // t = int
// double
double x1 = 10; // x1 = int double
var y1 = -123.2; // y1 = double
num z1 = -1.5; // z1 = double
var t1 = double. parse("-1.5");
// t1 = double

```

→ string ↔

```

String s1 = "test"; // s1 = string
var s2 = "test"; // s2 = string
var s3 = 'test';
// double quotes with escaped char
var s4 = "te/m/st";
// new string
var s5 = n"te/m/st";
// multi-line string

```

var  $\alpha$  = " " Multi-line  
 string " " ;  
 //  $\alpha$  char  $\leftrightarrow$  if var or \$expression  
 var  $x = 10$ ; var  $y = 20$ ;  
 var  $\alpha_1 = "x = \$x"$ ; //  $\alpha_1 = x = 10$   
 var  $\alpha_2 = "y = \$\{y\}"$ ; //  $\alpha_2 = y = 20$   
 var  $\alpha_3 = "\alpha_0m - \$\{y + y\}"$ ; //  $\alpha_3 = 20$   
 var  $\alpha_4 = \alpha "x = \$x"$ ; //  $x = \$x$   
 var  $\alpha_5 = "x = \$z"$ ; // error (compilation)

- toate tipurile sunt declarate din tipul dynamic  $\leftrightarrow$  date & variabile este declarata dynamic, își poate schimba tipul la runtim (or similar ca variabilele float tip din Python);
- dynamic  $\leftrightarrow$  dynamic  $d = 10$ ; //  $d = 10$   
 $d = "Test"$ ; //  $d = Test$   
 $d = true$ ;  
 $print(\&d)$ ; // true

- diferența dintre dynamic și var  $\leftrightarrow$  dynamic este un tip care impinge că nice variabilă / obiect pe care îl assignat / să nu obiect, iar var este var keyword ca cazăte că tipul variabili este înghesuit (dedus) din valoarea ei (dacă deducerea nu se poate face, tipul este considerat dynamic);

```

var v = 10;
print(v); // 10, v = int
v = "Test"; // compiler merge
var a; // a = dynamic
  
```

$a = 10; // 10$   
 $a = "Test"; // Test$   
 $a = true; // true$

→ operatorul  $\text{is} \Leftrightarrow$  verifică tipul că e deosebit de un anumit tip;

$\hookrightarrow$    
 int  $x = 10;$   
 $\text{print}(x \text{ is double}); // \text{true}$   
 $\text{print}(x \text{ is int}); // \text{true}$   
 $\text{print}(x \text{ is String}); // \text{false}$

## Operatori

- aritmetică ( $\Rightarrow$  binar:  $(+, -, *, /, \%, \sim)$ ) și
  - operatori ( $++, --$ ):
  - 1 renumerează rezultatul matematic al unei comparații (deci comparaționă  $\geq$  între 2 numere) și nu renumerează rezultatul sumei / produselor de unei comparații;
- var  $x = 10; \quad \text{var } y = 3; // 3.3333333$ 
  - $\text{print}(x/y); // 3$
  - $\text{print}(x \% y); // 1$
  - $\text{print}(x \sim y); //$  și
- bitwise ( $\Leftrightarrow$  binar:  $(\wedge, \vee, \wedge\wedge, \vee\vee, \leftarrow, \rightarrow)$ )
  - operatori ( $\wedge, \vee$ ) și shiftare la dreapta, mărgește ( $\ll$  și  $\gg$ ) și operează pe valori signed, iar operatorul  $\gg$  face pentru valori la dreapta pentru valori unsigned;
    - șiftrarea la dreapta pentru valori unsigned:  $=, +=, -=, *=, /=, \% =, \gg =, \ll =,$
    - șiftrarea la stânga pentru valori signed:  $=, +=, -=, *=, /=, \% =, \gg =, \ll =;$
  - operatori  $\gg\gg =, \sim\sim =, \& =, |=, \wedge =, \vee =, \wedge\wedge =, \vee\vee =;$

→ operator ?? = wegeert volgende expressie den  
deur dat deel expressie den waarde niet null;

↳ var x; // x = null

x ?? = 20;

print(x); // 20

var y = "Test"; // y = string

y ?? = "test";

print(y); // Test

so!

→ relationali  $\leftrightarrow >, >=, ==, !=, <, <=, \leq, \geq$ , so!  
(returneert een waarde baserend op  
vergelijkingen van twee nummer)

→ logici  $\leftrightarrow \text{true}, \text{false}, !$  (nu niet geforceerd op nummer)

as doors oor waarde baserend;

↳ var x = 10; var y = 3;

var z = x > y; // true

var d1 = (x > y) || (x == y); // true

→ conditionale:

1) condition ? valueIfTrue : valueIfFalse;



↳ var x = 10; var y = 3;

var z = x > y ? 2 : 3;

// x > y true  $\Rightarrow 2$

2) expression1 ?? expression2  $\leftrightarrow$  doet  
expression1 nu = null, danne resultaat  
van de expression1, anders expression2;

var x; // x = null

var z = 20;

var y = x ?? 10; // y = 10

var a = z ?? 200; // a = 200

→ sonderbare (subscript occurs):

- 1) object / array [index] ;  
2) object / array ? [index]  $\Leftrightarrow$  va gi true  
elementul corespondator de la indexul object / array  
nu este null , iar dacă este null , valoarea  
returnată va fi null (dacă ? [...] ] este  
goală în practică ar trebui să se exprime  
în valoarea returnată este null , ceea ce  
va fi ignorată);

var x; // null  
var y = x ? [1]; // y=null  
x ? [1] = 10; // x nu este schimbat

member access

- member access:  
1) object.member ;  
2) object?.member  $\leftrightarrow$  see if access  
member doesn't object me a null;  
- = "best";  
- ("to")); If true

dear daat og  
er  $\Rightarrow$  "test" 's  
point (ss. contains ("x")); // have  
ver +;  
point (t.. contains ("x")); // must  
- under must, value

↳ dacă obiect este null, valoarea  
atributelor e null;

operator cascading member

- return value

multiple objects

1) object .. member .. member .. member .. member

↳ dt access to multiple methods or data members of a class

class Teste { int x=10, int y=20; }

↳ void main()

var t = new Teste();

t .. x = 200;  
.. y = 400;  
print(t.x); print(t.y); // 200 400

(12)

2) object?.. member .. member .. member .. member  
↳ dé acces à multiple méthode ou  
dans membre a plusieurs object , donc  
object n'a pas de null (ce qui vaudrait  
que la méthode n'a pas de valeur  
dans member mais si ce n'a  
qui la ? member) :

→ opérateur cast  $\Leftrightarrow$  object as type (dans constante  
pas possible , codice va courir + exception) ;  
↳ var t = 1;  
var x = t as double;

### Strings

→ concaténation  $\Leftrightarrow$  string1 + string2 ;

↳ s = "Da" + "st"; // Dart

→ toString()  $\Leftrightarrow$  fonction qui transforme un  
objet en un string (génère un object à une);  
↳ var x = 10;  
print("x = " + x.toString()); // x=10

print("x=\$x"); // x=10

→ multiplication  $\Leftrightarrow$  string1 \* n Multiplication ;

↳ s = "Da" + "n" \* 3 + "t"; // Dart

→ opérateur []  $\Leftrightarrow$  accès au caractère de la string  
à l'index (string = all indexées comprisent  
au 0);

print("Dart"[0]); // a  
print("Dart"[20]); // exception

- .length ( $\leftrightarrow$  size-ul nume string);  
functie pentru a obtine lungimea string-ului
- .toLowerCase(), .toUpperCase() ( $\leftrightarrow$  functii pentru conversie (transformare) toate caracterele dintr-un string in litere mici, litere mari);
- .trim(), .trimLeft(), .trimRight() ( $\leftrightarrow$  functie pentru a revedea toate spatii, spatii din stanga, spatii din dreapta;
- var s = "[ Dart ]";  
present(s, trim()); // [Dart]  
present(s, trimLeft()); // [ Dart ]  
present(s, trimRight()); // [ Dart ]  
present(s, trim()); // [ Dart ]
- .substring(indexStart) ( $\leftrightarrow$  functie ce ia substringul de la indexStart pana la indexEnd);
- .substring(indexStart, indexEnd) ( $\leftrightarrow$  functie ce ia subtringul de la indexStart pana la indexEnd - 1 (indexEnd > indexStart, exceptie);
- var s = "Dart language";  
present(s, substring(5)); // language  
present(s, substring(3,10)); // t language  
present(s, substring(3,4)); // t  
present(s, substring(3,1)); // exception  
present(s, substring(3,1)); // exception
- .startsWith(stringToSearch), .contains(stringToSearch) ( $\leftrightarrow$  functie ce testeaza daca stringul respectiv este continut in stringToSearch, stringToSearch);
- .endsWith(stringToSearch, stringToSearch) ( $\leftrightarrow$  functie ce testeaza daca stringul respectiv este terminat cu stringToSearch;
- .substring(index, index) ( $\leftrightarrow$  nu este returnat nimic);

→ .startsWith (sToSearch, index), .contains (sToSearch, index) ↳ functie ce testează dacă stringul ~~contine~~ are sToSearch de la poziția index, ~~contine~~ sToSearch începând cu poziția index;

↳ var s = "Dant language";

```
1 true  
print(s.startsWith("Dant"));  
1 true  
print(s.startsWith("at", 2));  
1 false  
print(s.startsWith("at", 3));  
1 true  
print(s.endsWith("ge"));  
1 true  
print(s.endsWith("Dant language"));  
1 false  
print(s.endsWith("age"));  
1 true  
print(s.contains("t l"));  
1 false  
print(s.contains("tl"));  
1 false  
print(s.contains("Dant", 2));  
1 true  
print(s.contains("tl", 3));
```

→ .indexOf (sToSearch), .lastIndexOf (sToSearch) ↳ functie care căută primele, ultimele index unde se poate sToSearch în string și începând către de la poziția 0 și dacă se găsește, returnează indexul respectiv, altfel -1;

- .lastIndexOf(`toSearch`, `i`) , .indexOf(`toSearch`, `i`)
  - ↳ la jfl ce la valoare sought, din cadrul string de la poziția `i`, mai de la 0;
  - poziție de la poziția `i`, mai de la 0;
  - ↳ var `s` = "Dont language";
 

```
print(s.indexOf("at", 1)); //2
print(s.indexOf("at", 10)); //1
```
- .compareTo(`toCompare`) ↳ funcție a returnată 0, dacă `this` = `toCompare`, returnată 1, dacă `this` > `toCompare` și -1, altfel;
  - pozitiv = (⇒ returnată true/false)
  - funcție de egalitatea celor 2 stringuri;
  - ↳ print("abc" == "abc"); //true
 print("abc" > "abc"); //error (compile)
 print("abc".compareTo("abc")); //1
- .split(`toSplit`) ↳ funcție a separă stringul delimitat separatorul `toSplit` și returnând un array cu elementele după split;
  - var `s` = "R,G,B";
 

```
var l = s.split(',');
          l[0] = R, l[1] = G, l[2] = B
```
- .replaceAll(`toSearch`, `toReplace`) ↳ funcție ce schimbă toate aparițiile lui `toSearch` din string
  - cu `toReplace`;
  - replaceFirst(`st1, st2`) ↳ funcție ce schimbă prima apariție a lui `st1` din string cu `st2`, cindacei `st1` și `st2` sunt stringuri corespunzătoare de la poziția 0;
 

```
st1, st2, i) ↳ la jfl ce la
          lui st1 din string corespunzătoare de la poziția i;
          st1 și st2 sunt stringuri
```
  - replaceFirst(`st1, st2, i`) ↳ la jfl ce la
 `st1` și `st2` sunt stringuri
 cindacei `st1` și `st2` sunt stringuri
 începând de la poziția `i`;

→ replace Range (istart, iEnd, s) ( $\Rightarrow$  functie  
 a schimbă poziția dină string începând de  
 la poziția istart până la iEnd - 1 cu s;  
 ↳ var s = "D-at mega-missing";  
 ↳    // Dart programming  
 ↳    print(s.replaceRange(" ", "a"));  
 ↳    // Dart mega-missing  
 ↳    print(s.replaceFirst(" ", "a"));  
 ↳    // Dart programming  
 ↳    print(s.replaceFirst(" ", "a", 4));  
 ↳    // Dart mega-missing  
 ↳    print(s.replaceRange(0, 2, "DAA"));

→ String Buffer ( $\Leftarrow$  clasă cu care se poate crea  
 stringuri (ele mai importante și metode  
 sunt write, writeAll, clear);

↳ var sb = new StringBuffer();  
 ↳    sb.write("I like");  
 ↳    sb.write(" Dart");  
 ↳    print(sb.toString()); // I like Dart

### Control Flow Instructions

→ if(s)  
 {  
 1) if (condition) <then-part>;  
 2) if (condition) <then-part>  
 else <else-part>;

↳ condition trebuie să devină valori  
 booleane;

↳ `var a = 10; // a = const  
if (a) a += 1; // error`

$\rightarrow \text{while } \leftrightarrow \left\{ \begin{array}{l} \text{while (condition)} \\ \quad \langle \text{do - part} \rangle; \\ \text{break and si continutul set-ului de variabile si} \\ \text{condition trebuie sa contina valori} \\ \text{bunene,} \\ \text{continuare la injectarea lucru; } \\ \text{Sintaxa este } \text{do...while} \leftrightarrow \end{array} \right.$

$\rightarrow \text{do...while } \leftrightarrow \left\{ \begin{array}{l} \text{do } \{ \\ \quad \langle \text{do - part} \rangle \\ \quad \downarrow \text{while (condition);} \\ \text{break si continutul set-ului de variabile si condition} \\ \text{trebuie sa contina valori bune; } \\ \text{int } i=0, s=0; \\ \text{do } \{ \\ \quad i++; \\ \quad \text{if }(i>2 == 0) \text{ continue;} \\ \quad s+=i; \\ \quad \text{while }(i < 5); \\ \quad print(s); // s(1+3+5) \end{array} \right.$

$\rightarrow \text{for } \leftrightarrow \left\{ \begin{array}{l} \text{for (initialization; condition; increment)} \\ \quad \{ \\ \quad \quad \quad \dots \}; \\ \text{break si continutul set-ului de variabile si teste} \\ \text{trebuie sa contina 3 segmente din paranteze sunt} \\ \text{optional; } \\ \text{var } s=0, i=0; \\ \text{for } (i;i<5) \{ \\ \quad \text{if }(i>2 == 0) \text{ break;} \\ \quad s+=i; \\ \quad i++; \} \quad \text{if } s == 100 \end{array} \right.$

- for (for loop) este o operatie care utilizeaza mai multe variabile ( $\leftrightarrow$  la fiecare iteratie sunt 3 componente, variabile)
  - var si separate prin  $>;$
  - $\hookrightarrow \text{for } (i=0, j=5; i+j < 30; i++, j++)$
- for (este similar cu  $\text{for } i = 0 \dots n - 1$  "like Dart")
  - var  $s = ""$  si  $s = s + \text{split}(w, ",")$
  - $\text{for } (var w \in s)$
  - $\quad \text{print}(w);$
  - $\hookrightarrow$

$\rightarrow \text{switch} \leftrightarrow \left\{ \begin{array}{l} \text{switch (expression)} \\ \quad \text{case value1}: \dots \\ \quad \text{case value2}: \dots \\ \quad \dots \\ \end{array} \right.$

$\hookrightarrow \text{var } x = 1;$   
 $\text{switch } (x) \{$   
 $\quad \text{case 1: print("one"); break;}$   
 $\quad \text{case 2: print("two"); break;}$   
 $\quad \text{default: print("a"); break;}$   
 $\}$  || one

- break nu poate fi omis, doar un singur though este necesar (de exemplu, doar la exemplul precedent omisi break la case 1, sau si print si case);
- ↪ solutie este sa se scrie lui continuu catre un label de la start de o valoare case;

$\hookrightarrow \text{var } x = 1;$   
 $\text{switch } (x) \{$   
 $\quad \text{case 1: print("1"); continue case-2;}$   
 $\quad \text{case-2:}$   
 $\quad \quad \text{case 2: print("2"); break;}$   
 $\quad \quad \text{default: print("0"); break;}$   
 $\}$

- continue ne de posibilitatea si de a da skip direct la un caz degetat separat (ne respecta la urmatorul caz);
  - astfel pot fi folosite constante switch;
  - var color = "Red";
    - switch (color) {
      - case "Red": print("R"); break;
      - case "Blue": print("B"); break;
      - case "Green": print("G"); break;
- ↓ // R

### Custom Types

- enumas  $\leftrightarrow$  {
    - 1) numar (names) 1val, val2, ... valn;
    - 2) numar  $\leftrightarrow$  numele unei valori val1;
    - 3) index  $\leftrightarrow$  indexul unei valori val1;
  - enum good {apple, orange, carrot};
    - void main() {
      - var a = good.apple;
      - print(a.name); // apple
      - print(a.index); // 0
  - for (var i in good.values) {
    - print(i.toString());
    - // good.apple good.orange good.carrot
    - print(i.index.toString());
    - // 0 1 2
- ↓
- un enum nu poate avea un amanunt tip;

- ↳ `enum Good; const {a, b, c} // values`
- enumereaza multe valori relate cu o valoare numerică specifică asociată;
- ↳ `enum Good {a, b = 5, c} // values`
- enumerează multe valori și este să o valoare să fie;
- ↳ `const i = Good::a; // values`
- `typedefs` ↔ `typedefs <name> = existing-type;`
- ↳ `typedef int64 = int;`  
`void main() {`  
 `int64 x = 10;`  
 `print(x); // 10`

- ### Functions
- `forma basică` ↔ `return-type <function-name>(<parameters>)`  
 $\left\{ \begin{array}{l} \dots \\ [return value] \end{array} \right.$
  - ↳ `int sum(int x, int y) {`  
 `return x + y;`
  - `return-type prototype` și `return` ↔ se definește tipul de return al statmentului de return, din tipul de return al expresiei
  - `forma simplificată` ↔ `return-type <function-name>(<parameters>) ⇒ expression;`

↳ `int sum(int x, int y) = x+y;`

→ parametrii  $\Leftrightarrow$  1) pozitionali;

2) optionali (intre [...] );

3) named (intre {---});

→ sunt doar si parametrii optionali;

→ o functie poate avea (nicio data comanda);  
si named (nu trebuie sa fie obligatoriu)

si named (nu trebuie sa fie obligatoriu)

debut cu pozitionali si named trebuie definiti;

dupa ce pozitionali, daca exista;

required pentru parametru named  $\Leftrightarrow$  sunt

obligatori;

↳ 1) positional and optional parameters  
`void get(int x, [int y=10]);`

2) optional parameters with default values  
`void get1([int x=20, int y=10]);`

3) void get2([x=20, y=10]);

→ functii cu parametrii optionali sunt o valoare  
defautl pot fi folosite de ce? este folosit debut  
tip;

↳ type?  $\Leftrightarrow$  variabila este si ~~este~~ o valoare  
tip, si null (valoarea default nu este  
necesara pentru ca deea variabila nu este  
gesitata, este setata ca null);

↳ `int sum([int? x, int? y]) {`  
 `if ((x is int) && (y is int)) {`  
 `return x + y;`  
 `} else {`  
 `return 0;`

↳ `void main() {`  
 `print(sum(10, 20)); // 30`  
 `print(sum(10)); // 10`  
 `print(sum()); // 0 }`

→ parametrii numai trebuie specificati cu numele care sunt utilizati;

↳ int sum (int base, int x=10, int y=20);  
 $\Rightarrow \text{base} + x + y;$

```
void main()
{
    print(sum(10, x: 5)); // 10 + 5 + 20 = 35
    // 10 + 10 + 10 = 30
    print(sum(10, y: 10));
    // 10 + 10 + 20 = 40
    print(sum(10));
    // 10 + 5 + 3 = 18
    print(sum(10, y: 5, x: 3));
    // error
    print(sum(10, z: 3));
}
```

→ ce și de parametrii optionali, valoarea default poate fi omisă, dacă nu este specificată?

↳ void fct (int? x, required int y = 10);

→ lambdas  $\Leftrightarrow \begin{cases} 1) (\text{parameters}) \rightarrow \dots \\ 2) (\text{parameters}) \rightarrow \text{expression} \end{cases}$

↳ var sum = (int x, int y)  $\rightarrow$  return x + y;;  
 var mul = (int x, int y)  $\rightarrow$  x \* y;;

```
var sum = (int x, int y)  $\rightarrow$  return x + y;
var mul = (int x, int y)  $\rightarrow$  x * y;
print(sum(10, 20)); // 30
print(mul(5, 3)); // 15
```

→ o funcție poate returna un lambda (o funcție anonimă), folosind keywordul function ca să nu se returneze (dacă lambda - și generează parametrii de return, îl va împiedica de utilizare de către parametrii sau valori locale);

chiar dacă funcția respectivă nu a returnat);

↳ Funcționare get (înainte de)

$$\text{var } m = n + 1;$$

return (înainte de)  $\Rightarrow \% m = 0$ ;

↳

```
void main() {  
    var g = get(); // m = 8  
    print(g(24)); // true
```

↳

→ funcții generice  $\Leftrightarrow$  nu își tip specifică;

↳ sum(x, y)  $\rightarrow$  x + y;

void main() {

var v = sum(10, 20);

print(v.sumType); // int

print(sum("test", "abc").sumType);

// String

var v3 = sum(1.5, 10);

print(v3.sumType); // double

↳

→ funcții generice pot conține multe parametrii definite, cu un tip (închidând tipul de return) și astfel, definiții generice (important să fie operație și să fie realizată);

↳ sum(x, int y)  $\rightarrow$  x + y;

void main() {

print(sum(40, 20)); // 200

print(sum("ab", 3)); // ababab

↳

→ funcții generice pot returna tipuri diferențiate;

```

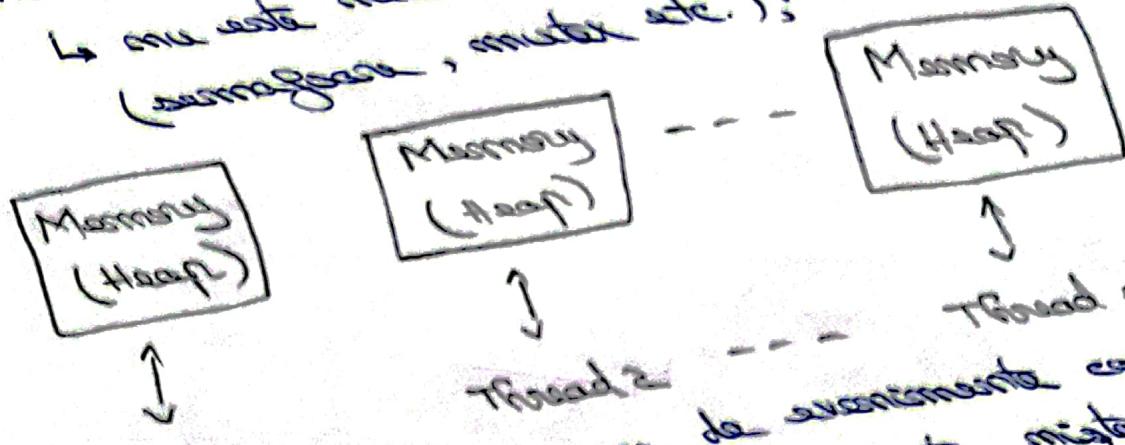
    ↳ foo (int x) {
        if (x > 100) return "result";
        else return tree;
    }

```

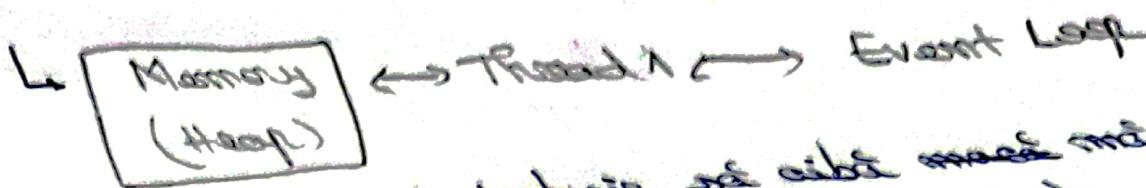
↳ acelasi rezultat se obtine folosind dynamic ca tip de return;

### Threading Model

- memoria nu este unica / aceasi pentru toate fiile de executie, ci fiecare procesor are o memorie proprie de executie, fiecare cu memoria lui propriu (un heap propriu)  $\leftrightarrow$  cea a procesorului care l-a creat (un heap proprietar)  $\leftrightarrow$  cea disponibila pentru acest thread
- memoria este unica / obiecte de sincronizare threadurile;
- sau este memoria de mijlocie / obiecte de sincronizare (semifese, mutex etc.);



- există o buclă de procesare de evenimente care să aplice la fiecare thread care are astăzi niste evenimente dintr-o echipă și le procesează  $\Rightarrow$  nu mai există o nevoie de cod care trebuie să fie sănătățial sau săptămânal, ci fiecare thread are un eveniment de evenimente de cod care trebuie să fie sănătățial sau săptămânal, care procesează niste evenimente;
- isolată  $\leftrightarrow$  memoria, threadul și evenimentul;



- fiecare program care trebuie să aibă acces la el
  - un isolate (el care reprezintă o mașină);
- se pot trimite mesaje / evenimente în event loop
  - ↪ modul de comunicare dintre un isolate și un altul;
- variabile / obiecte dintr-un isolate nu sunt disponibile pentru un alt isolate;

### Future Object

- un obiect care atunci cand este creat nu conține nicio valoare, ci doar stie ce fel de valoare va avea în următoarele momente și nu poate să cunoască valoarea sa așteptătă până la un moment de timp;
  - ↪ obiectul variabilă/obiectul a fost creat, există însă și o întrebare ce obiect este o variabilă variabilă și cum să rezolve declararea unui număr de obiecte pînă când acest lucru nu este gata;
- pentru a rezolva această întrebare de timp ( $t_1, \dots, t_n$ ) nu se întâmplă nimic (avantajul este că nu trebuie să precizezi momentele), iar după ce vom reprezenta întrebarea de către o astfel de formă, avem cănd obiectul este întâmpinat și cănd obiectul variabilă este completă, și cănd obiectul variabilă este definit și cănd callback-ului este apelat (nu se vor apela callback-urile și pînă când obiectul variabilă este completă și cănd obiectul variabilă este definit și cănd callback-ului este apelat).

- static Future<out template> -> Future<T>;
- compilazione → causa = essere, dare o class forward
- estende / implementare con class mix parte e class Future<T>;
- trovi importante libreria "dark:async";

```

    ↳ int get() {
        print("Future code called");
        return 10;
    }

```

```

    ↳ void main() {
        print("start main code");
        Future<int> delayed(
            FutureObj = Future<int>.delayed(
                Duration.seconds(2),
                get));
        print("End of main code");
    }

```

↳  
 // Start main code  
 // FutureObj created  
 // value will come in 2 seconds  
 // and then get will run => return 10  
 // End main code  
 // after 2 seconds => get is called  
 // Future code called  
 // get returns 10

- gliere Future all' < metode then (prendere un  
primo un callback / o di mettendo end un  
Future not complete) => trovi se prima che un  
callback per fare di processare valenza per fare un  
callback

print - s;

```
↳ void processValue (int value) {
    print ("Value received is $1 value");
}

J
void main () {
    print ("Start main code");
    var futureObj = Future<int>.delayed (
        Duration.seconds(2),
        () => 10);
    futureObj.then (processValue);
    print ("End main code");
}

J
// Start main code
// End main code
// Value received is 10
```

→ print then return back some Future - val;

```
↳ Future<int> first (int value) {
    print ("First $1 value");
    return Future<int>.delayed (
        Duration.seconds(value),
        () => 2).then (second);
```

```
J
Future<int> second (int value) {
    print ("Second $1 value");
    return 0;
}

void main () {
    print ("Start");
    Future<int>.delayed (
        Duration.seconds(2),
        () => 4).then (first);
    print ("First");
    Future<int>.delayed (
        Duration.seconds(2),
        () => 2).then (second);
}

J
// Start, First -> 4, Second -> 2
```

→ Future<T>.value  $\leftrightarrow$  de constructor găzduiește o  
 returnare a valoarei;  
 ↳ tot mereu se procesează imediat;  
 ↳ Future<int>.value(10).then(  
 (value)  $\Rightarrow$  print(value));

→ genetării și punerea obiectelor Future;  
 ↳ int runCode() {  
 point("Run code");  
 throw "exception";  
 }

↳ int onErrorCallback(error) {  
 point("Error", \$error);  
 return 0;

↳ void main() {  
 Future<int>.sync(runCode)  
 .catchError(onErrorCallback)  
 .then((value)  $\Rightarrow$  print(value))  
 .whenComplete(() =>  
 point("Done"));

↳  
 // Run code  
 // Error : exception  
 // 0  
 // Done

### Timer!

- un obiect care funcționează ca un eveniment care este repetat după o perioadă specifică de timp;
- construcția și proprietăți;

```

↳ main() {
    point("start");
    Timer(Duration(seconds: 2),
        () => point("triggered timer"));
    point("end");
}

```

↳  
 // start  
 // end  
 // triggered timer ← after 2 seconds

```

void timerCallback(Timer t) {

```

↳ (t.tick >= 4)

t.cancel();

point("Timer called : tick = \${t.tick}");

```

}
main() {
    point("start");
}

```

Timer.periodic(

Duration(seconds: 2,

periodicCallback);

point("end");

↳  
 // every 2 seconds, timercallback is called

// start

// end

// Timer called : tick = 1

// Timer called : tick = 2

// Timer called : tick = 3

// Timer called : tick = 4

↳  
 // constructor simple Timer este triggered deea &

este triggered deea &  
 date, iar Timer.periodic, ce merge, este triggered  
 periodic (s date la durata date ca parametru);

## Synchronisation

- begrijpbaar await of anyone :
  - 1) anyone ( $\Rightarrow$  trebuie asteptat la finalul execuției funcției) la un moment dat se execută același lucru, execuția se poate întârzi și la un moment dat acest lucru se va întâmpla după ce se va contineaza de același mod și să înceapă o altă funcție;
  - 2) await ( $\Rightarrow$  trebuie să se joace unui Future) cu un metode care returnează un Future, care poate să devină sănătoșă, să rămână ca în stare de execuție sau să fie completată și să fie executată încă din nou, să se repete o altă funcție sau să se continue să se joace și să fie completată și să fie executată încă din nou;

→ funcțile anyone trebuie să aibă tipul de return

Future ( $\Leftarrow$ ) sau void;

$\hookrightarrow$  void main() anyone {  
 print("start");  
 var f = Future<void>.delayed(  
 Duration(seconds: 5),  
 () => print("5 seconds later"));  
 f.then((value) =>  
 print("5 seconds later"));  
}

await f;  
 print("end");

5 start, 5, end

→ a priori aussi futur que valeur renvoyée de future (var < name> = await < future > copie la valeur renvoyée de future en name) ;

↳ void main() async {  
 print("start");  
 var f = Future<int>.value(123);  
 var result = await f;  
 print(result);  
 print("end");  
}  
// start  
// 123  
// end

→ ce code si await futur face à une la mai melle oblige Future pour venir linked ;

↳ void main() async {  
 int x = await Future<int>.value(10);  
 print(await Future<int>.value(++x)); // ou

→ try --- catch --- try --- catch --- finally fait si besoins frontier variable dont un Future ou une valeur, et ce à cause/exception ;  
généralement lorsque, se fait une Future-ness ;  
→ pour gérer,

↳ Future<int>.get(int val) → comme  
 => val + val;  
 main() + print("start");  
 get(10).then((val) => print(val));  
 print("end");

// start, end, 100

## Streams

- un obiect de tipul Future care poate sa devină o secvență de date sau valori (nu este obligatoriu să nu este mutabil și să fie crește) (peste funcție care este indiferent);
- listeners  $\leftrightarrow$  callbacks care sunt notificate de fiecare dată când un obiect este disponibil;
- listeners pentru Streamuri  $\leftrightarrow$  .then pentru Future;
- template  $\leftrightarrow$  Stream<T> (similar cu List<T>);
- generic Stream este o metodă listener care primește ca parametru o funcție care void onData(T event)
- ca să primește valoarea obiectului de tip Stream și că să ar trebui să oprimem, rezultatul (tipul de return) de sunte sistemul să îndepărteze .cancel(), StreamSubscription<T> (permite .cancel(), .suspend(), .resume());
- Stream simple  $\leftrightarrow$  un singur listener;
- Stream broadcast  $\leftrightarrow$  mai mulți listeners;
- Stream<T>. periodic  $\leftrightarrow$  constructor pentru a crea un Stream (funcția va fi apelată periodic);
  - ↳ main() + Stream<Int>. periodic(Duration(seconds: 2), (event) => count++))
  - ↳ listen((event) => print(event)), .listen((event) => print(event)),
  - ↳ Stream<Int>. periodic(Duration(seconds: 2), (event) => print(event)))

1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99, 101, 103, 105, 107, 109, 111, 113, 115, 117, 119, 121, 123, 125, 127, 129, 131, 133, 135, 137, 139, 141, 143, 145, 147, 149, 151, 153, 155, 157, 159, 161, 163, 165, 167, 169, 171, 173, 175, 177, 179, 181, 183, 185, 187, 189, 191, 193, 195, 197, 199, 201, 203, 205, 207, 209, 211, 213, 215, 217, 219, 221, 223, 225, 227, 229, 231, 233, 235, 237, 239, 241, 243, 245, 247, 249, 251, 253, 255, 257, 259, 261, 263, 265, 267, 269, 271, 273, 275, 277, 279, 281, 283, 285, 287, 289, 291, 293, 295, 297, 299, 301, 303, 305, 307, 309, 311, 313, 315, 317, 319, 321, 323, 325, 327, 329, 331, 333, 335, 337, 339, 341, 343, 345, 347, 349, 351, 353, 355, 357, 359, 361, 363, 365, 367, 369, 371, 373, 375, 377, 379, 381, 383, 385, 387, 389, 391, 393, 395, 397, 399, 401, 403, 405, 407, 409, 411, 413, 415, 417, 419, 421, 423, 425, 427, 429, 431, 433, 435, 437, 439, 441, 443, 445, 447, 449, 451, 453, 455, 457, 459, 461, 463, 465, 467, 469, 471, 473, 475, 477, 479, 481, 483, 485, 487, 489, 491, 493, 495, 497, 499, 501, 503, 505, 507, 509, 511, 513, 515, 517, 519, 521, 523, 525, 527, 529, 531, 533, 535, 537, 539, 541, 543, 545, 547, 549, 551, 553, 555, 557, 559, 561, 563, 565, 567, 569, 571, 573, 575, 577, 579, 581, 583, 585, 587, 589, 591, 593, 595, 597, 599, 601, 603, 605, 607, 609, 611, 613, 615, 617, 619, 621, 623, 625, 627, 629, 631, 633, 635, 637, 639, 641, 643, 645, 647, 649, 651, 653, 655, 657, 659, 661, 663, 665, 667, 669, 671, 673, 675, 677, 679, 681, 683, 685, 687, 689, 691, 693, 695, 697, 699, 701, 703, 705, 707, 709, 711, 713, 715, 717, 719, 721, 723, 725, 727, 729, 731, 733, 735, 737, 739, 741, 743, 745, 747, 749, 751, 753, 755, 757, 759, 761, 763, 765, 767, 769, 771, 773, 775, 777, 779, 781, 783, 785, 787, 789, 791, 793, 795, 797, 799, 801, 803, 805, 807, 809, 811, 813, 815, 817, 819, 821, 823, 825, 827, 829, 831, 833, 835, 837, 839, 841, 843, 845, 847, 849, 851, 853, 855, 857, 859, 861, 863, 865, 867, 869, 871, 873, 875, 877, 879, 881, 883, 885, 887, 889, 891, 893, 895, 897, 899, 901, 903, 905, 907, 909, 911, 913, 915, 917, 919, 921, 923, 925, 927, 929, 931, 933, 935, 937, 939, 941, 943, 945, 947, 949, 951, 953, 955, 957, 959, 961, 963, 965, 967, 969, 971, 973, 975, 977, 979, 981, 983, 985, 987, 989, 991, 993, 995, 997, 999, 1001, 1003, 1005, 1007, 1009, 1011, 1013, 1015, 1017, 1019, 1021, 1023, 1025, 1027, 1029, 1031, 1033, 1035, 1037, 1039, 1041, 1043, 1045, 1047, 1049, 1051, 1053, 1055, 1057, 1059, 1061, 1063, 1065, 1067, 1069, 1071, 1073, 1075, 1077, 1079, 1081, 1083, 1085, 1087, 1089, 1091, 1093, 1095, 1097, 1099, 1101, 1103, 1105, 1107, 1109, 1111, 1113, 1115, 1117, 1119, 1121, 1123, 1125, 1127, 1129, 1131, 1133, 1135, 1137, 1139, 1141, 1143, 1145, 1147, 1149, 1151, 1153, 1155, 1157, 1159, 1161, 1163, 1165, 1167, 1169, 1171, 1173, 1175, 1177, 1179, 1181, 1183, 1185, 1187, 1189, 1191, 1193, 1195, 1197, 1199, 1201, 1203, 1205, 1207, 1209, 1211, 1213, 1215, 1217, 1219, 1221, 1223, 1225, 1227, 1229, 1231, 1233, 1235, 1237, 1239, 1241, 1243, 1245, 1247, 1249, 1251, 1253, 1255, 1257, 1259, 1261, 1263, 1265, 1267, 1269, 1271, 1273, 1275, 1277, 1279, 1281, 1283, 1285, 1287, 1289, 1291, 1293, 1295, 1297, 1299, 1301, 1303, 1305, 1307, 1309, 1311, 1313, 1315, 1317, 1319, 1321, 1323, 1325, 1327, 1329, 1331, 1333, 1335, 1337, 1339, 1341, 1343, 1345, 1347, 1349, 1351, 1353, 1355, 1357, 1359, 1361, 1363, 1365, 1367, 1369, 1371, 1373, 1375, 1377, 1379, 1381, 1383, 1385, 1387, 1389, 1391, 1393, 1395, 1397, 1399, 1401, 1403, 1405, 1407, 1409, 1411, 1413, 1415, 1417, 1419, 1421, 1423, 1425, 1427, 1429, 1431, 1433, 1435, 1437, 1439, 1441, 1443, 1445, 1447, 1449, 1451, 1453, 1455, 1457, 1459, 1461, 1463, 1465, 1467, 1469, 1471, 1473, 1475, 1477, 1479, 1481, 1483, 1485, 1487, 1489, 1491, 1493, 1495, 1497, 1499, 1501, 1503, 1505, 1507, 1509, 1511, 1513, 1515, 1517, 1519, 1521, 1523, 1525, 1527, 1529, 1531, 1533, 1535, 1537, 1539, 1541, 1543, 1545, 1547, 1549, 1551, 1553, 1555, 1557, 1559, 1561, 1563, 1565, 1567, 1569, 1571, 1573, 1575, 1577, 1579, 1581, 1583, 1585, 1587, 1589, 1591, 1593, 1595, 1597, 1599, 1601, 1603, 1605, 1607, 1609, 1611, 1613, 1615, 1617, 1619, 1621, 1623, 1625, 1627, 1629, 1631, 1633, 1635, 1637, 1639, 1641, 1643, 1645, 1647, 1649, 1651, 1653, 1655, 1657, 1659, 1661, 1663, 1665, 1667, 1669, 1671, 1673, 1675, 1677, 1679, 1681, 1683, 1685, 1687, 1689, 1691, 1693, 1695, 1697, 1699, 1701, 1703, 1705, 1707, 1709, 1711, 1713, 1715, 1717, 1719, 1721, 1723, 1725, 1727, 1729, 1731, 1733, 1735, 1737, 1739, 1741, 1743, 1745, 1747, 1749, 1751, 1753, 1755, 1757, 1759, 1761, 1763, 1765, 1767, 1769, 1771, 1773, 1775, 1777, 1779, 1781, 1783, 1785, 1787, 1789, 1791, 1793, 1795, 1797, 1799, 1801, 1803, 1805, 1807, 1809, 1811, 1813, 1815, 1817, 1819, 1821, 1823, 1825, 1827, 1829, 1831, 1833, 1835, 1837, 1839, 1841, 1843, 1845, 1847, 1849, 1851, 1853, 1855, 1857, 1859, 1861, 1863, 1865, 1867, 1869, 1871, 1873, 1875, 1877, 1879, 1881, 1883, 1885, 1887, 1889, 1891, 1893, 1895, 1897, 1899, 1901, 1903, 1905, 1907, 1909, 1911, 1913, 1915, 1917, 1919, 1921, 1923, 1925, 1927, 1929, 1931, 1933, 1935, 1937, 1939, 1941, 1943, 1945, 1947, 1949, 1951, 1953, 1955, 1957, 1959, 1961, 1963, 1965, 1967, 1969, 1971, 1973, 1975, 1977, 1979, 1981, 1983, 1985, 1987, 1989, 1991, 1993, 1995, 1997, 1999, 2001, 2003, 2005, 2007, 2009, 2011, 2013, 2015, 2017, 2019, 2021, 2023, 2025, 2027, 2029, 2031, 2033, 2035, 2037, 2039, 2041, 2043, 2045, 2047, 2049, 2051, 2053, 2055, 2057, 2059, 2061, 2063, 2065, 2067, 2069, 2071, 2073, 2075, 2077, 2079, 2081, 2083, 2085, 2087, 2089, 2091, 2093, 2095, 2097, 2099, 2101, 2103, 2105, 2107, 2109, 2111, 2113, 2115, 2117, 2119, 2121, 2123, 2125, 2127, 2129, 2131, 2133, 2135, 2137, 2139, 2141, 2143, 2145, 2147, 2149, 2151, 2153, 2155, 2157, 2159, 2161, 2163, 2165, 2167, 2169, 2171, 2173, 2175, 2177, 2179, 2181, 2183, 2185, 2187, 2189, 2191, 2193, 2195, 2197, 2199, 2201, 2203, 2205, 2207, 2209, 2211, 2213, 2215, 2217, 2219, 2221, 2223, 2225, 2227, 2229, 2231, 2233, 2235, 2237, 2239, 2241, 2243, 2245, 2247, 2249, 2251, 2253, 2255, 2257, 2259, 2261, 2263, 2265, 2267, 2269, 2271, 2273, 2275, 2277, 2279, 2281, 2283, 2285, 2287, 2289, 2291, 2293, 2295, 2297, 2299, 2301, 2303, 2305, 2307, 2309, 2311, 2313, 2315, 2317, 2319, 2321, 2323, 2325, 2327, 2329, 2331, 2333, 2335, 2337, 2339, 2341, 2343, 2345, 2347, 2349, 2351, 2353, 2355, 2357, 2359, 2361, 2363, 2365, 2367, 2369, 2371, 2373, 2375, 2377, 2379, 2381, 2383, 2385, 2387, 2389, 2391, 2393, 2395, 2397, 2399, 2401, 2403, 2405, 2407, 2409, 2411, 2413, 2415, 2417, 2419, 2421, 2423, 2425, 2427, 2429, 2431, 2433, 2435, 2437, 2439, 2441, 2443, 2445, 2447, 2449, 2451, 2453, 2455, 2457, 2459, 2461, 2463, 2465, 2467, 2469, 2471, 2473, 2475, 2477, 2479, 2481, 2483, 2485, 2487, 2489, 2491, 2493, 2495, 2497, 2499, 2501, 2503, 2505, 2507, 2509, 2511, 2513, 2515, 2517, 2519, 2521, 2523, 2525, 2527, 2529, 2531, 2533, 2535, 2537, 2539, 2541, 2543, 2545, 2547, 2549, 2551, 2553, 2555, 2557, 2559, 2561, 2563, 2565, 2567, 2569, 2571, 2573, 2575, 2577, 2579, 2581, 2583, 2585, 2587, 2589, 2591, 2593, 2595, 2597, 2599, 2601, 2603, 2605, 2607, 2609, 2611, 2613, 2615, 2617, 2619, 2621, 2623, 2625, 2627, 2629, 2631, 2633, 2635, 2637, 2639, 2641, 2643, 2645, 2647, 2649, 2651, 2653, 2655, 2657, 2659, 2661, 2663, 2665, 2667, 2669, 2671, 2673, 2675, 2677, 2679, 2681, 2683, 2685, 2687, 2689, 2691, 2693, 2695, 2697, 2699, 2701, 2703, 2705, 2707, 2709, 2711, 2713, 2715, 2717, 2719, 2721, 2723, 2725, 2727, 2729, 2731, 2733, 2735, 2737, 2739, 2741, 2743, 2745, 2747, 2749, 2751, 2753, 2755, 2757, 2759, 2761, 2763, 2765, 2767, 2769, 2771, 2773, 2775, 2777, 2779, 2781, 2783, 2785, 2787, 2789, 2791, 2793, 2795, 2797, 2799, 2801, 2803, 2805, 2807, 2809, 2811, 2813, 2815, 2817, 2819, 2821, 2823, 2825, 2827, 2829, 2831, 2833, 2835, 2837, 2839, 2841, 2843, 2845, 2847, 2849, 2851, 2853, 2855, 2857, 2859, 2861, 2863, 2865, 2867, 2869, 2871, 2873, 2875, 2877, 2879, 2881, 2883, 2885, 2887, 2889, 2891, 2893, 2895, 2897, 2899, 2901, 2903, 2905, 2907, 2909, 2911, 2913, 2915, 2917, 2919, 2921, 2923, 2925, 2927, 2929, 2931, 2933, 2935, 2937, 2939, 2941, 2943, 2945, 2947, 2949, 2951, 2953, 2955, 2957, 2959, 2961, 2963, 2965, 2967, 2969, 2971, 2973, 2975, 2977, 2979, 2981, 2983, 2985, 2987, 2989, 2991, 2993, 2995, 2997, 2999, 3001, 3003, 3005, 3007, 3009, 3011, 3013, 3015, 3017, 3019, 3021, 3023, 3025, 3027, 3029, 3031, 3033, 3035, 3037, 3039, 3041, 3043, 3045, 3047, 3049, 3051, 3053, 3055, 3057, 3059, 3061, 3063, 3065, 3067, 3069, 3071, 3073, 3075, 3077, 3079, 3081, 3083, 3085, 3087, 3089, 3091, 3093, 3095, 3097, 3099, 3101, 3103, 3105, 3107, 3109, 3111, 3113, 3115, 3117, 3119, 3121, 3123, 3125, 3127, 3129, 3131, 3133, 3135, 3137, 3139, 3141, 3143, 3145, 3147, 3149, 3151, 3153, 3155, 3157, 3159, 3161, 3163, 3165, 3167, 3169, 3171, 3173, 3175, 3177, 3179, 3181, 3183, 3185, 3187, 3189, 3191, 3193, 3195, 3197, 3199, 3201, 3203, 3205, 3207, 3209, 3211, 3213, 3215, 3217, 3219, 3221, 3223, 3225, 3227, 3229, 3231, 3233, 3235, 3237, 3239, 3241, 3243, 3245, 3247, 3249, 3251, 3253, 3255, 3257, 3259, 3261, 3263, 3265, 3267, 3269, 3271, 3273, 3275, 3277, 3279, 3281, 3283, 3285, 3287, 3289, 3291, 3293, 3295, 3297, 3299, 3301, 3303, 3305, 3307, 3309, 3311, 3313, 3315, 3317, 3319, 3321, 3323, 3325, 3327, 3329, 3331, 3333, 3335, 3337, 3339, 3341, 3343, 3345, 3347, 3349, 3351, 3353, 3355, 3357, 3359, 3361, 3363, 3365, 3367, 3369, 3371, 3373, 3375, 3377, 3379, 3381, 3383, 3385, 3387, 3389, 3391, 3393, 3395, 3397, 3399, 3401, 3403, 3405, 3407, 3409, 3411, 3413, 3415, 3417, 3419, 3421, 3423, 3425, 3427, 3429, 3431, 3433, 3435, 3437, 3439, 3441, 3443, 3445, 3447, 3449, 3451, 3453, 3455, 3457, 3459, 3461, 3463, 3465, 3467, 3469, 3471, 3473, 3475, 3477, 3479, 3481, 3483, 3485, 3487, 3489, 3491, 3493, 3495, 3497, 3499, 3501, 3503, 3505, 3507, 3509, 3511, 3513, 3515, 3517, 3519, 3521, 3523, 3525, 3527, 3529, 3531, 3533, 3535, 3537, 3539, 3541, 3543, 3545, 3547, 3549, 3551, 3553, 3555, 3557, 3559, 3561, 3563, 3565, 3567, 3569, 3571, 3573, 3575, 3577, 3579, 3581, 3583, 3585, 3587, 3589, 3591, 3593, 3595, 3597, 3599, 3601, 3603, 3605, 3607, 3609, 3611, 3613, 3615, 3617, 3619, 3621, 3623, 3625, 3627, 3629, 3631, 3633, 3635, 3637, 3639, 3641, 3643, 3645, 3647, 3649, 3651, 3653, 3655, 3657, 3659, 3661, 3663, 3665, 3667, 3669, 3671, 3673, 3675, 3677, 3679, 3681, 3683, 3685, 3687, 3689, 3691, 3693, 3695, 3697, 3699, 3701, 3703, 3705, 3707, 3709, 3711, 3713, 3715, 3717, 3719, 3721, 3723, 3725, 3727, 3729, 3731, 3733, 3735, 3

↳ `sent compute (sent current)`  
↳ `(current > 5) throws "exception";`  
`return current + 2;`

↳ `main()`  
`Stream<int>.periodic(Duration records: 2),`  
`compute)`  
• `listen((event) => print(event))`  
`onError: (err) => print(err))`

↳ `exception`  
|| 0, 2, 4, 6, 8, 10, exception

→ cancellation: falle la listener ( $\Rightarrow$  chiede stop  
specie user request, tot mi spieghi Streamable, ci  
aspettiamo un action mai deposta;  
metodicità quindi Streamable ha terminato;

↳ `main()`  
`Stream<int>.fromIterable([1, 2, 3, 4])`  
• `listen((e) => print(e))`  
`onError: () => print("error"))`

↳ `Stream from an Iterable`  
|| 0 fromIterable  $\leftrightarrow$  Stream from an Iterable  
|| 1, 2, 3, 4, Done

→ functii prim eare un Stream cu: record si  
returnand un alt Stream  $\leftrightarrow$ . map<U>, . skip,  
. repeat, . take, . where, . skipWhile, . takeWhile  
(toate au ca tip de return Stream<T>);

→ constructor  $\leftrightarrow$  .fromFuture , .quantifiers  
(primă ca parametru un storable de Future)  
· storable , .principie , .value ;

↳ main()  
stream < int>.fromStorable([1,2,3,4])

.map((val)  $\Rightarrow$  val + 2)

.distinct((val)  $\Rightarrow$  print(val));

5  
|| 2,4,6,8

main()  
stream < int>.fromIterable([1,2,3,4,5,6,7,8,9,10])  
.map((val)  $\Rightarrow$  val + 3)  
.where((val)  $\Rightarrow$  val % 4 == 0)  
.distinct((val)  $\Rightarrow$  print(val));

11-12,24

→ Stream<T> Stream<T>.distinct  $\leftrightarrow$  funcție ce  
separe elementele consecutive care sunt similare  
(nu are primă ca parametru  $\Rightarrow$  funcție care arată  
când 2 elemente sunt egale, by default / poate  
acest parametru este să se utilizeze un operatorul  
 $= =$ )

↳ main()  
stream < int>.fromIterable([1,2,3,3,3,4,5,5,5])

.distinct()

.distinct((val)  $\Rightarrow$  print(val));

5  
|| 1,2,3,4,5

```
main() {  
    Stream<int> s = fromIterable([1, 2, 3, 1, 2, 3]);  
    • distinct()  
    • listen((val) => print(val));  
}  
// 1, 2, 3, 1, 2, 3
```

```
Stream<int> s = fromIterable([1, 3, 5, 8, 10, 200]);  
• distinct((x1, x2) =>  
    x1 < x2 == x2 < x1);  
• listen((val) => print(val));  
}  
// 1, 3, 5, 8, 10, 200
```

// 1, 2, 3

→ alle Funktionen von Stream sind  $\Leftarrow$  .contains ,  
• elementAt , .any , .every , .lastWhere ,  
• lastWhere ( teste returniert ein Fiktivwert );

```
main() {  
    point("start");  
    var g = Stream<int> s = fromIterable([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]);  
    • contains(x),  
    g.then((val) => print(val));  
}  
// start, end, true
```

→ .contains returniert true / false , .or .lastWhere ,  
.lastWhere returniert gewisse , sie .lastWhere Element  
core respects = condition date point -> Funktion ;

```
main() {  
    Stream<int> s = fromIterable([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]);  
    • lastWhere((x) => x == 7);  
    • then((val) => print(val));  
}  
// 7
```

→ In .firstWhere of .lastWhere exists preconditions  
where  $\leftrightarrow$  Stream must be a terminal of one  
arrow or queue for future of an on-the-fly function  
because to have a view on future, does  
one can just observe

↳ main() {  
 Stream<int>.fromIterable([1, 2, 3, 4, 5, 6])  
 .firstWhere((v) => v == 11,  
 orElse: () => 100)  
 .then((val) => print(val));  
}  
// 100

→ .any, .every ( $\leftrightarrow$ ) means an element, toate  
elementale respects a condition;  
Stream<int>.fromIterable([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])  
.any((v) => v % 2 == 0)  
.every((v) => print("\$1 v\$"))  
odd numbers

// true odd numbers  
Stream<int>.fromIterable([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])  
.every((v) => v % 2 == 0)  
.then((v) => print("\$\_1 v\$\_2"))  
all odd numbers: \$1 v2\$  
all odd numbers: false

→ generic, runtime or static prints-in stream ( $\leftrightarrow$ )  
.forEach, .join, .reduce (return value object Future);  
.forEach of .list must similar;  
↳ Stream<int>.fromIterable([1, 2, 3, 4, 5])  
.forEach((v) => print(v));  
↳ .join ( $\leftrightarrow$ ) is toate elementale, to print static-in

String și fișiere puncta - un separator de către parametrii:

→ reduce ( $\Rightarrow$  primește 2 elemente consecutive din Stream și le face să facă o operare standard

Stream și este să fie un alt element de tipul  $T$  și din către un alt element de tipul  $T$ :

↳ Stream<int>.fromIterable([1, 2, 3, 4, 5])

· join("1")

· then((val)  $\Rightarrow$  point(val)):

|| 1 2 3 4 5

point(await Stream<int>.fromIterable([1, 2, 3, 4, 5]))

· reduce((-r, e)  $\Rightarrow$  r + e);

|| 1 5 (1+2+3+4+5)

→ StreamController<T>  $\Leftrightarrow$  obiect prin care putem să furnizăm date și primește o serie de callbacks (onListen(), onPause(), onResume(), onCancel()) în cazul acțiunii de interacție cu stream. Acestea se execută, este să pună stări de proprietate, onCancel, onListen, onPause, onResume, stream;

→ stream  $\Leftrightarrow$  adunător un stream în cazul introducă date (conform unei reguli);

→ metodele unei StreamController  $\Leftrightarrow$  all (a excepție valoare), .addError, .close();

↳ main() { var sc = StreamController<int>();

Timer.periodic(Duration(seconds: 1, (timer) {

sc.send("Send \$) timer.tick \* 2 ));

sc.add(timer.tick \* 2);

});

sc.close();

sc.point("stream was closed"); } );

sc.stream.listen((val)  $\Rightarrow$  point(val)); }

|| Send 2, 3, Send 4, 4, Send 6, 6  
|| Send 8, 8 Stream was closed, 8

→ function generator ~~function~~: → stream;

↳ bool isPrime (int val) {  
 for (var i = 2; i < val / 2; i++)  
 if (val % i == 0) return false;  
 return true;  
}

• Iterable < int > getPrimes (int max) {  
 for (var i = 2; i < max; i++)  
 if (isPrime(i)) yield i;  
 return;  
}

↓  
main() {  
 const i = getPrimes(10);  
 console.log(i);  
 // 2, 3, 5, 7  
}

→ acle iterator maakt een deel;

↳ acle iterator maakt een deel;

→ function generator ~~function~~ return value  
Stream < T > kan los de Iterable < T >, en  
async + kan los de async + xi anders in capsule  
function existé yield value van yield await

Future < value >;

↳ Future < bool > isPrime (int val) {  
 for (var i = 2; i < val / 2; i++)  
 if (val % i == 0) return Future.value(false);  
 return Future.value(true);  
}

Stream <int> getNa (int max) const \* {  
 for (var i = 2; i < max; i++) {  
 var result = await instance(i);  
 eg (result) yield i;  
 }  
}

main() {  
 getNa(10).listen((val) => print(val));  
}  
// 2, 3, 5, 7

→ await for  $\Leftrightarrow$  modelliert spät bereit System ca  
diesen Elementen denkt ein  $\Rightarrow$  Stream ist die Struktur  
(Asynchron API und  $\Rightarrow$  all previous tests elements  
dien Stream);  
↳ pointer o return o value obtained from  
processing elementen denkt ein Stream;  
await for as terminal and stream as terminal  $\Rightarrow$

→ await for as terminal and stream as terminal my code  
the bus of even generic of stream  $\Rightarrow$  Stream <int> sum(Stream <int>) const \* {  
 as independent (as terminal);  
}

↳ Future <int> sum(Stream <int>) const \* {  
 var sum = 0;  
 await for (int value int =>)  
 sum += value;  
 return sum;  
}

main() {  
 print(await sum(sum(Stream <int>.  
 point(await sum(Stream <int>.  
 .getNa([1, 2, 3, 4])))));  
}

// 10  
// without await sum => 0

→ metoda er await for  $\leftrightarrow$  stream synchronous parte un stream;

### Transformator

- StreamTransformer  $\langle T_1, T_2 \rangle \leftrightarrow$   $\text{func} \circ \text{transform}$  a unui stream de la un emit tip ( $T_1$ ) in alt tip  $T_2$
- Stream  $\langle T_1 \rangle$  Stream  $\langle T_1 \rangle \cdot \text{transform} \leftrightarrow$  metoda a unei transformator de la un emit tip ( $T_1$ ) in alt Stream;
- Stream  $\langle T_1 \rangle$  Stream  $\langle T_2 \rangle$  Stream  $\langle T_1 \rangle \cdot \text{transformer}$  generata un obiect StreamTransformer care transforma un Stream in alt Stream;
- StreamTransformer  $\cdot$  bind  $\leftrightarrow$  generare de fereastra date bind un Stream in fereastra de un alt Stream (punete un Stream de input si un Stream de output);
- StreamTransformer are si o metoda statica castFrom;

↳ class Unique implements StreamTransformer<int, int>  
↳ var - et = StreamController<int>();  
↳ var - set = set<int>();  
@ override  
StreamTransformer<RS, RT> cast<RS, RTX>();  
actual StreamTransformer <RS, RT> castFrom(Third);

@ overrides  
stream<int> bind(Stream<Set> stream) {  
 stream.list((val) {  
 if (!set.contains(val)) == false) {  
 set.add(val);  
 }  
 }  
 return et.stream();  
}

```

main() {
    Stream<int> stream = ...;
    stream |> printIterable([1, 2, 3, 1, 2, 3])
        .transform(unique())
        .listen((val) => print(val));
}

```

J 11, 2, 3

→ Donc on n'a pas de transformation en bibliothèque

'donc: convert':

```

main() {
    var idx = 0;
    File("a") < path to a file > |
        openRead().transform(AsciiDecoder())
        .transform(Linesplitter())
        .listen((line) =>
            print(`${idx} ${line}`));
    idx++;
}

```

→ dupl. transform se peut utiliser via fil de  
fonction applicable pour Stream API (map, where etc)

### Templates / Generics

→ accéder à une partie de cod pour éviter réécriture, des ce 1er  
tipeur de date (permettre à writer d'utiliser cette valeur);

→ class class-meme < T1, [T2, --, Tn] > { -- };

→ class MyTemplate < T, G > {

```

    T obj1;
    G obj2;
    MyTemplate::init(T obj1, G obj2);
    obj1 = obj2;
    void print() {
        print("obj1 = " + obj1.toString());
        print("obj2 = " + obj2.toString());
    }
}

```

```
main() { var m = MyTemplate<int, String>.  
    .init(10, "Test"); }
```

```
m.print();  
// obj1 = 10, obj2 = Test
```

↓

→ T extends <base-type>  $\Rightarrow$  + greater if done on tip  
derivative from base-type;

↳ class MyTemplate <T extends num> {

```
T sum(T obj1, T obj2) {
```

return (obj1 + obj2) as T;

// without & extends num,

// compilation error

// + not defined for object?

↓

```
main() { var m = MyTemplate <int>();  
print(m.sum(10, 20)); // 30
```

// without extends num, the compiler

// doesn't have the guarantee

// the + operation can happen

↓

→ base-type part of & is often about (definite definition);

↳ abstract class PrintInterface {

```
void print();
```

```
class MyTemplate <T extends PrintInterface> {
```

```
void print(T obj) => obj?.print();
```

↳ class MyInt extends PrintInterface {

```
int i;
```

```
MyInt(this.i);
```

(48)

@ override

void print() -> print(i); }

main() { var m = MyTemplate < MyInt >();  
m.print(MyInt(10)); // 10 }

→ templatebazaan en goede oplossing is de enkel de  
methode:

↳ class MyString { String data = "5";  
void from < T > (T obj) => data = obj.toString(); }

main() { var m = MyString();  
m.from < int > (10); // m.data = 10  
m.from < double > (1.23); // m.data = 1.23 }

## Libraries

→ import "src";  
→ import "src" as alias;  
→ import "src" where <component>;  
→ import "src" hide <component>;  
→ src ↪ { 1) don't: <name> (de exemplu don't: id),  
2) package: <path>;  
3) <path> (găzduie local);  
→ object File constructori ↪ File (string path),  
File fromRawPath (List<List path>);  
File fromUri (List<List path>);  
Random nextInt (val);  
for 'don't: math' => Random.nextInt (val);  
some sort random func for val; }

- operăriile cu fiziere i/o  $\leftrightarrow$  scrierea sau citirea (în "dant: ss");
  - funcții ~~pentru~~ fiziere ( $\leftrightarrow$  File.readAsLineSync), File.writeAsStringSync (string text), File.writeAsBytesSync ( $\langle$  buffer) etc;
  - File.writeAsBytesSync ("b.bim").writeAsBytesSync ()
  - ↳ File("b.bim").writeAsBytesSync ()
  - ↳ int [1, 2, 3, 4, 5];
  - $\langle$  int  $\rangle$  ca să scriem.
- funcții asociate pe fiziere;
  - de la fizice sunt nume;
  - de la custom ( $\leftrightarrow$  numerele sunt altă fizieră)
- librărie custom ( $\leftrightarrow$  când avem nevoie de ea, fizice import ("fileName.dart"));
  - și fizice și -varName sunt considerate "private"
  - și nu sunt accesibile în alte fiziere sau altă fizică
  - de la librărie / fiziere unde sunt definite;
  - În cadrul unei fizieri, fizice nu pot fi accesate, deoarece sunt în scopul -varName
- și clasele și fizice sunt de tip "private";
- numărul - în fizice sunt numerei (ca la variabile);
- fizice unde numărul este import "fileName.dart" ca `<name>`; (elementele din fizice sunt fizice și fizice)

### Regular Expressions

- RegExr (string path, {bool multiline = false, bool caseSensitive = true, bool unicode = false, bool dotAll = false});  $\leftrightarrow$  obiect pentru a crea o expresie regulară;

- RegExP ·
- Funktion  $\Leftrightarrow$  Standard < RegExp Methods > \ all Methods ,  
 RegExpMatch? , RegExp . startMatch , best  
 RegExp . hasMatch , Match? , RegExp . matchAsPrefix ,  
 RegExp . stringMatch ;  
 String RegExp . stringMatch ;  
 String RegExp . stringMatch now ;
  - im general want flexible stringMatch now ;
  - ↪ var s = RegExp( $a "[0-9]^+$ ");  
 point(s . hasMatch("Hello 1234")); // true  
 // 1234  
 point(s . stringMatch("Hello 1234 world"));  
 point(s . matchAsPrefix("Hello 1234 world"));  
 point(s . matchAsPrefix("Hello 1234  
 world", 6));  
 var res = s . matches("Hello 1234 world", 6);
  - reg(s, ! = success)  
 reg("c", 10) " \$ 1 res . start(), \$ 1 res . end() ",  
 point(" \$ 1 res . start(), \$ 1 res . end() ");  
 var s = "Hello world";  
 var res = "Hello" .  
 reg(s, ! = "Hello" .  
 s . allMatches());  
 reg(s, ! = "Hello"  
 "0,5" .> "Hello"  
 "1,1" .> "world"  
 point(" \$ 1 res . start(), \$ 1 res . end() ");  
 \$ 2 res . group(0) );  
 \$ 2 res . split(regExp(a"1s+"))  
 reg(var s, ! = success)  
 "Hello"  
 "1234"  
 "Dark"  
 point(s);

- Funktion den String case sensitive von object Pattern  
 si nicht gelöst express regular ( $\Leftrightarrow$ ) . split , . contains ,  
 startsWith , . indexOf , . lastIndexOf , . replaceAll ,

- replaceFirst , - replaceAll Mapped , - split MapJoin ,
- replaceFirst Mapped ;

↳ Pattern  $\Leftrightarrow$  clé de base case grata se  
aibit 2 generate : o expresie regulată  
sau un string;

↳ var a = RegExp(a "[0-9]+");  
var s = " 100 KILOS OF APPLES COST  
25 DOLLARS ";  
print(s.replace(a, "<NUMBER>");  
print(s.replace(a, "less of apples cost  
1<NUMBER> dollars"));  
print(s.replace(a, "1w+"));  
a = RegExp(a "1w+");  
" 100 KILOS OF APPLES COST  
25 DOLLARS "  
print(s.replaceMapped(a,  
"(m) => m[0].toUpperCase()  
?? "" ));

$\rightarrow$  group[0]  $\Leftrightarrow$  tot stringul care a facut match (mai  
sunt stringuri mai mici);  
 $\rightarrow$  .groupCount  $\Leftrightarrow$  nr. de grupuri (de exemplu:  
sunt 127, 5, 5 sunt 3 grupuri: 127, 5, 5, 5,  
pentru o expresie regulată care punte mai multe nr.  
separata prin .);  
↳ nr. total de grupuri este .groupCount + 1  
pentru ca [0] este tot tot stringul;  
↳ caud string pentru prima grupuri sunt  
de la i=0, cu i < .groupCount;

## Date & Time

```
var d = DateTime.now(); // yyyy-mm-dd  
// HH:mm:ss.milliseconds  
d = DateTime(1900, 1, 1, 12, 30, 45);  
print(d); // 1900-01-01 12:30:45.000  
d = DateTime.fromMicrosecondsSinceEpoch(1000);  
print(d); // 1970-01-01 02:00:00.000000  
print(d.year); // 1970  
print(d.day); // 1  
print(d.weekday); // 14
```

### ↳ DateTime constructor:

→ operatii / functii pentru date / timp  $\leftrightarrow$  DateTime

DateTime.add, DateTime.subtract, DateTime.compare, bool DateTime.isAfter, bool DateTime.isBefore, DateTime.isSameMomentAs,

DateTime.toUtc;

Duration  $\leftrightarrow$  parametrii numai pentru si, oră etc.

durată este calculată ca suma parametrilor;

↳ constructor de tip const;

get proprietate Duration  $\leftrightarrow$  .inDays, .inHours, .inMinutes, .inSeconds, .inMilliseconds;

.inMicroseconds (returnării sunt);

↳ var d1 = DateTime(2000, 1, 1, 12, 30);  
var duration = Duration(days: 3, minutes: 10);  
var d2 = d1.add(duration);  
print(d1); // 2000-01-01 12:30:00.000  
print(d2); // 2000-01-04 12:40:00.000  
print(d1.compareTo(d2)); // -1  
print(d1.compareTo(d2)); // 1

point (de difference (d), en Minutes); 11433c

- functii pentru a parcurge si a obtine un DateTime corespondator formatului  $\leftrightarrow$
- parcurge, .tryParse (returneaza DateTime);
  - ↳ DateTime::parse ("2000-05-10 12:30");
- datele dintr-un format de tipul "2000-05-10" sau "2000 05/10";
- Startwatch  $\leftrightarrow$  obiect pentru a măsura timpul;
  - 1) metode  $\leftrightarrow$  .start, .stop, .reset (returneaza void, nu are parametri);
  - 2) proprietati  $\leftrightarrow$  .Elapsed (returnează Duration), .Elapsed Microseconds, .Elapsed Milliseconds, .isRunning;

### Claselor

- Object  $\leftrightarrow$  clasa de baza din care implicit toate clasele sunt derive;• classe clasă cu metă proprietăți/metode sunt finalizate, Type metodaType, dynamic noSuchMethod (Invocation  $\leftrightarrow$ ) și string toString();
- o clasa conține  $\leftrightarrow$  unde se mai multe constructori, operatori, getere și setere (pe care le proprietăți), date membrii metode;
- nu este nevoie să existe un destructor;

## Data Members

- methods de a defini = date members:
  - ↳ var  $v1[1, v2, v3, \dots, v_m]$ ;
  - ↳ types  $v1[1, v2, v3, \dots, v_m]$ ;
  - ↳ type of  $v1$   $v = <values>$ ;
- dacă  $v1$  = date member sau i se dat valoare, să avă tipul dynamic (se este  $v1$ ):
  - ↳ class Test { var t;  
main() { var t = Test();  
print(t.x); // printType(); // null  
print(t.y); // printType(); // int  
t.x = 10; // t.x.printType() = int  
t.x = "test"; // t.x.printType() = String } }
- în general nu putem crea o variabilă și să nu-i initializezi;
  - ↳ int x; // error = 'int' doesn't allow null initialization;
- dacă nu vom să initializez după tip (va fi null, dacă nu putem face)?
  - este initializez;
  - ↳ int? x; // null implicit
- date member read-only ⇒ ca final;
  - ↳ class Test { final int x = 30;  
main() { var t = Test();  
print(t.x); // final int x = 30;  
print(t.y); // null  
print(t.z); // null  
t.x = 40; // error } }
- când se crează o variabilă final, se crează deoarece

- general, nu si setim;
- sistem cu variabile statice  $\Rightarrow$  ele vor fi accesate  
clasei, nu sunt constante (mai multe constante,  
vor avea acces la ele și ele vor fi comune);
- variabile statice pot fi accesate, deoarece  
numele clasei și numele - o constantă;
- ↳ class Test { static int x = 30; }  
    main() { print(Test.x), // 30  
          var t = Test();  
          print(t.x); // error }
- variabile statice constante  $\Rightarrow$  ce statice const;
- nu există conceptul de public / private / protected
- (totul este considerat ca fiind accesibile), dar există  
un mod de a restricta accesul la numele  
debutului  $\Leftrightarrow$  ce - în găsi numele variabilei,  
metodei etc.;
- ↳ static int x = 30;
- operațional? peste și speriat când accesăm o date  
membru / un camp dintr-o clasa  $\Leftrightarrow$  se dă acces  
la acel camp, dacă obiectul chiar există;
- ↳ class Test { static int x = 30; }  
    main() { Test? t; // null object  
          t?.x = 10; }

### Construcții

- nu putem avea mai multe metode cu același nume, dar  
parametrii diferiți  $\Rightarrow$  sistem cu un singur constructor
- named (class\_name([v1, v2, ..., vn]));
- putem avea multe named constructor și



`<class-name.function-name>([v1, v2, ..., vm]);` if

- un string argument;
- data-member va fi serie <tip> d<sub>1</sub>, ..., d<sub>m</sub>;  
(declarare cu un tip + val, dar nu initializare);
- (de la constructor implicit după initializare), deacă la initializare implicită, după constructor ( $\text{d}_1 = \text{value}_1$  = <value>);

`<data-member-name> {`

`class <class-name> {`  
 `<data-type> d1, d2, ..., dm;`  
 `<class-name> ([v1, v2, ..., vm]): d1 = value1,`  
 `d2 = value2, ..., dm = valuem. } -- }`

`class Test { int x, y;`  
 `Test(): x=10, y=20 } }`

`main() { var t = Test();`

`print(x, y); // 10`

`print(x, y); // 20 } }`

- return of initializare = variabila de la input  
⇒ se așteaptă initializare ca mai sus în constructor  
(constructorul și va rămâne valoarea initializată);

`class Test { int x=1;`  
 `Test(): x=2 } }`

`main() { var t = Test(); print(x); print(x); // 2 }`

- o variabilă nu se poate inițializa în cadrul constructorului (se poate inițializa doar ce mai sus);

`Test() { x=10; } // error (int x;)`

↳ Test() { int x, y; } Test(this.x, this.y);

`this;`  
 `class Test { int x, y; } Test(this.x, this.y);`  
 `main() { var t = Test(10, 20); print(x, y); print(x, y); // 10 }`

- date constructorul este gata (15), return punem deasupra ; la sfârșit ;
  - ↳ `class Test { int x,y; Test(): x=10, y=20; }`
- nu putem să inițializăm o date membru cu o altă date membru în constructor ;
  - ↳ `class Test { int x,y; Test(int value): y=value, x=this.y + this.y; }`
- nested constructors sunt și obiecte statice temporale
  - ↳ `class Test { int x,y; Test.empty(): x=0, y=0; }  
 Test.withX(int val): x=val, y=0;  
 Test.withXY(int x, int y);  
 Test.withT(Test t): t=t; //t.x=t.y=0  
 main() { var t = Test.empty(); t.x=10, t.y=20  
 t = Test.withX(10); t.x=10, t.y=0  
 t = Test.withXY(1,2); t.x=1, t.y=2; }`
- un constructor poate apela un alt constructor ;
  - ↳ `class Test { int x,y; Test(int val): x=val, y=val; }  
 Test.empty(): this(0);`
- redirectioning constructors nu sunt de același (ctor1 → ctor2 → ctor3, ctor3 → ctor1  $\Leftrightarrow$  nu poate fi)
  - ↳ `ctor2 → ctor3, ctor3 → ctor1;`
  - ↳ constructorii obiectelor care să creze obiecte constante  $\Rightarrow$  ca const în gata constructorului (dacă)

pe care le aducă constructorul trebuie să fie același;

↳ class Test { float x, y;  
const Test. constructorObj(): x=0, y=0,;  
main() { var t = Test. constructorObj(); } }

→ clasa singleton ( $\Rightarrow$  un singur obiect);  
→ numed constructor static ( $\Leftrightarrow$  ca factory);

↳ class Singleton { static Singleton obj = Singleton();  
int x, y;  
Singleton(float val): x=val, y=val;  
factory Singleton. sameObj(): return obj;  
main() { var t1 = Singleton. sameObj();  
var t2 = Singleton. sameObj();  
print(t1 == t2); // true } }

### Properties

→ proprietăți care să cumere și înțele date membre;

→ getter  $\Leftrightarrow$  <data-type> get {var-name>} ; return value;

→ setter  $\Leftrightarrow$  set(<data-type> value) { } ;  
implicit există un getter și un setter pentru fiecare date membru, mai puțin, dacă (setările nu sunt de setat);

↳ class Grades { float m, e;  
Grades(this.m, this.e);  
isn't get average() {  
return (m+e)/2; // division by 2 }  
set average(float val) { m = e = val } }

```

main() {
    int st = Grades(10, 2);
    print(st. average); // 9
    st. average = 10;
    print(st. m), // 10
    print(st. s); // 10
}

```

- proprietăți read-only ( $\Rightarrow$  definim doar gettere (de exemplu la exemplul precedent doar mă avem și settere,  $st. average = 10$ ; și și doar setare);
- gettere și settere sunt cele patru variabile accessible din alte biblioteci;
- ↳ class Test {
 int x = 0;
 set x(int value)  $\Rightarrow$   $x = value + 2$ ;
 int get x()  $\Rightarrow$   $x$ ;
}
main() {
 Test t = Test();
 t.x = 5;
 print(x.t); // 7
}
- operatorul  $++$  are efect pe proprietăți (de exemplu  $t. i++;$ ) → gettere și settere vor fi considerate triggered (mai sunt gettere și după settere);

### Methods

- <data-type> <fct-name> ([v<sub>1</sub>, ..., v<sub>n</sub>]) { ... } sau  
 <data-type> <fct-name> ([v<sub>1</sub>, ..., v<sub>n</sub>])  $\Rightarrow$  <return-type>;
- metodele nu suportă overloading ( $\Leftrightarrow$  nu putem avea mai multe funcții cu același nume și parametrii deosebiți  $\Rightarrow$  putem folosi variabile de pe altă numără pentru a evita asta);

```

    ↳ class Test { int x=3;
        int Add (int v1, {int? v2}) =>
            v2!=null ? x+v1+v2 : x+v1;
        bool isEven () { return x%2==0; }
    }

    main() { var t = Test();
        print(t.Add(5)); // 14
        print(t.Add(6,7)); // 18
        print(t.isEven()); // true
    }

```

- méthodes pas de suppression (override) => ~~obligatoire~~
- ② override une autre fonction ou une obligation
  - (de exemple system suppression toString())
- méthode dynamic (échec dynamique) :
  - fonction type dynamic (échec dynamique) :

```

    ↳ class Test { int x;
        Test(this.x);
        void m1Method (Invocation i) {
            if (i.isMethod) {
                if (i.memberName ==
                    "toString").contains("write"))
                    print("x = $x");
            }
        }
    }

    main() { dynamic t = Test(10);
        t.write(); // x=10
        var t1 = Test(10); t1.write(); // error
    }

```
- méthode pas de static => cette accède pas
  - à une classe (ne = system accès prints-e
 instances);

```

↳ class Test { const x = 10;
    static String getName() => "Test class";
    main() { print(Test.getName()); // Test class
    var t = Test(); t.getName();
}

```

### Operators

→ <data-type> operator <op-type> ([v<sub>1</sub>, ..., v<sub>n</sub>]) --->;

→ operators supported => <>, >, <=, >=, ==, +, -, \*, /, %, ~!, ^, t, , !, ~, >>, >>>, <<, [ ], [ ] =;

→ operators [] => assignment and value de  
la un anumit index dintr-un vector etc.;

↳ class Number { const x;
 Number(this.x);
 Number operator+(Number n) =>
 Number(x + n.x);
 Number operator\*(Number n) =>
 Number(x \* n.x);
 main() { var m = Number(10);
 var n2 = m1 + Number(20);
 print(n2.x); //30
}

→ operatorii sunt tot multe genetă, => nu suportă  
overloading (același operator cu parametrii diferenți),  
dar o soluție este să definim parametrii dynamic.

↳ Number operator+(dynamic val) {
 if (val is Number) return Number(x +
 (val as Number).x);
 if (val is int) return Number(x +
 (val as int));
 return Number(x);
}

## Inheritance

- deriveare  $\Rightarrow$  cu extends ;
  - interface  $\Rightarrow$  cu implements ;
  - class (nume) extends <base-class>
    - implements <interface>,  $\rightarrow$   $\downarrow \vee$  ;  
implementă funcție dintr-o altă clasă ;
  - mixin  $\Leftrightarrow$  formă de a implementa funcție dintr-o altă clasă ;
- ```
class Base { int x=10; }  
int GetX() { return x; }  
  
class Derived extends Base { int y=20; }  
main() { var d = new Derived();  
print("$d.x", "$d.y", "$d.GetX()"); // 10, 20, 10  
}
```
- metodele și datele membrele pot fi exprimate la derivare și putem accesa datele din clasa de bază cu super (la o proprietate / metodă) ;
  - class Base { int x=10; }  
class Derived extends Base { int x=20; }  
int GetBaseX() { return super.x; }  
main() { var d = new Derived();  
print("\$d.x", "\$d.GetBaseX()"); // 20, 10  
}
- dacă exprimăm = date membru , trebuie să fie de același tip ca cea din clasa de bază ;
  - în expresiile metodelor se spică accesul prin principiu că la datele membru  $\Leftrightarrow$  se va face une funcție din clasa derivată pentru un obiect din clasa derivată și funcția trebuie să aibă același nume , accesul generetic și accesul tip de return ca în clasa de bază ;

→ fiecare metoda dintr-o clasa este virtuala;  
 ↳ clasa B și int sp (int x, int y) =>  $x+y$ ; ↳  
 clasa D extinde B; ↳  
 @ override  
 int sp (int x, int y) =>  $x+y$ ; ↳  
 int sp (B b, int x, int y) {  
 int sp (B b, int x, int y) {  
 return b.sp(x, y); }  
 main() { var d = D();  
 print (sp(d, 3, 4)); //  $12 = 3+4$  } }

→ nu putem pune final înaintea unei clase  
 (pentru a evita ca nu se mai poate衍isa) și  
 trebuie să punem;

→ clasa abstractă => se pot defini metode genetice  
 echivalente cu cele virtuale puse din C++ care  
 nu au implementare (implementarea este săn  
 căză să o derivăze pe cea abstractă) și sunt  
 clasă și abstractă;

↳ abstract class Form {  
 string getName(); // abstract method  
 bool isForm(); }  
 class Circle extends Form {  
 // must override getName()  
 @ override  
 string getName() => "Circle"; } }

→ nu este posibil să folosești keywordul abstract <=>  
 clasele metodele abstracte trebuie să aibă și implementare;  
 → putem avea și proprietăți abstracte;

```

    ↳ abstract class Form { string get name => "Form"; }

    int x = 10; ↳
    ↳ class Circle extends Form {
        @override
        string get name => "circle"; ↳
    }

```

→ o clasa care poate implementa datele membru  
dintre o interfață (deși metodele și proprietatile);

```

    ↳ abstract class Form { int x = 10; }

    ↳ abstract class Form { abstract property }

    string get name; //abstract property ↳
    ↳ class Circle implements Form {
        //won't get the data members
        @override
        string get name => "circle"; ↳
        main() { var c = Circle();
        print("${c.x}"); //error } ↳
    }

```

→ o clasa poate implementa mai multe interfațe,  
(clase abstracte) ⇒ trebuie să suprascrie toate  
metodele / proprietatile abstracte din toate interfațele

→ dacă nu ~~nu~~ abstract în faza unei clase,  
proprietatile trebuie să să implementeze (să  
fie implementate și proprietatile ~~și funcțiile~~);

→ dacă în interfață, se acceștă metode cu parametri  
disponibile și este implementată, nu pot fi suprascrise

↳ amândouă metode;

↳ dacă interfață ar acceza funcție cu accezași  
parametri, suprascriere funcțională;

→ numai prin keywordul abstract putem avea  
metode / proprietăți, abstracte într-o clasa, altfel

implementarea este necesară și tot să nu extinde / implementează (chiar dacă clasa nu e tipică abstractă) :

→ key wordul covariant poate fi aplicat pentru a face astă parametrizare și generarea unei metode virtuale / abstracte atunci când sunt derivate din clasa de bază;

↳ class Form { }  
class Rectangle extends Form { }

abstract class Calc { }  
int compute (covariant Form f);  
class AreaCalc implements Calc { }

@ override  
int compute (Rectangle r) => 0;

|| without covariant => error

→ putem avea o clată care să extindă sau și implementația unei ;  
→ cand implementăm o clată , toate metodele și proprietățile trebuie implementate , chiar dacă nu sunt doar implementate în clasa de bază ;  
↳ mixin ( $\Rightarrow$  să nu fie metoda din clasa de bază și să utilizez ocazional cum suntem )

↳ mixin Test {  
int sum (int x, int y) => x+y; }  
class my with Test { }  
main() { var z = my();  
print(z.sum(10,20)); //30 }

→ se poate folosi clasă înlocuitor de mixin ;



- ↳ cu mixin, clasa nu poate fi instanciată;
- la mixin putem face overriding la funcție (funcționă ca și cea să fie funcție abstractă);
- mixin nu poate avea multe sau același număr de mixin, mixiny - - -
- la mixin se face în calcul și datele membru (ca la extends);
- ce extends putem extinde și singură clase;
- ce mixinurile multiple conțină ordinea →
- la mixinurile același variabilă de exemplu, se va da cea din urmă de la următoarele mixin;
- în valoarea de la următoarele mixin care se referă la o date membru avea 2 mixinuri care se referă la aceeași variabilă numai și tipuri diferențiate (⇒ scope);
- keywordul on ⇒ aplicarea unui mixin deasupra anumite clase (deivate din ea);
- ↳ class Number {
 mixin Sum on Number {
 sum(x,y) = x+y;
 }
 test extends Number with Sum {
 class test1 with sum {
 // ...
 }
 }
}

|                      | extends           | implements                  | with                                   |
|----------------------|-------------------|-----------------------------|----------------------------------------|
| Data members         | Yes               | No                          | Yes                                    |
| Methods              | Can be overridden | MUST be overridden          | Can be overridden<br>can be overridden |
| Properties           | Can be overridden | MUST be overridden          | Can be overridden<br>can be overridden |
| Abstract methods     |                   | Virtual, MUST be overridden |                                        |
| Abstract properties  |                   | Virtual, MUST be overridden |                                        |
| Multiple inheritance | No                | Yes                         | Yes                                    |

## Iterable

- interface (classe abstrakte);
- nu se poate crea obiecte Iterable, doar există metode proprietăți pe care orice obiect iterable le are  $\hookrightarrow T$  Iterable< $T$ >. \_\_init\_\_,  $\in$  Iterable< $T$ >. \_\_len\_\_,  $\in$  Iterable< $T$ >. \_\_iter\_\_, \_\_contains\_\_ , \_\_eq\_\_ Iterable< $T$ >. \_\_length\_hint\_\_, \_\_iterable< $T$ >. \_\_empty\_\_, \_\_bool\_\_ Iterable< $T$ >. isNotEmpty, base Iterable< $T$ >.
- .single returnează elementul care există într-un singur element care există în cale există un singur element în cale conținut (sau o excepție în caz contrar);
- metoda de c创are  $\hookrightarrow$  Set< $T$ > Iterable< $T$ >. \_\_setitem\_\_(), List< $T$ > Iterable< $T$ >. \_\_setitem\_\_(), Iterable< $V$ > Iterable< $T$ >. \_\_setitem\_\_() (obține un iterable testă toate elementele care există deja în c创ință și adaugă elementul curent, sau elementele existente de la c创ință);  
+  $\forall V \in \text{loc de } T$ ;
- metoda iterable  $\hookrightarrow$  Iterable< $T$ >. skip, Iterable< $T$ >. skipwhile, Iterable< $T$ >. take, Iterable< $T$ >. takewhile, Iterable< $T$ >. \_\_skip\_if\_cannot (tace returnările Iterable< $T$ >, skip și skip\_if\_cannot au ca parametru un întreg și scrie o funcție);  
+  $\forall$   $\lambda$  parametru  $\hookrightarrow$  Iterable< $V$ > Iterable< $T$ >. map< $T$ >, Iterable< $T$ >. expand, \_\_bool\_\_ Iterable< $T$ >. any, Iterable< $T$ >. reduce, \_\_str\_\_ Iterable< $T$ >. every, Iterable< $T$ >. \_\_join;
- string Iterable ca parametru un string (separătoare), sau scrie o funcție;  
 $\hookrightarrow$  Join permite să se scrie unul (separătoare), să se scrie o funcție;
- metoda pentru a itera prin elementele unei containere  $\hookrightarrow$  void Iterable< $T$ >. \_\_for\_each\_\_ (permite ca parametru o funcție, să de proceseze o elementelor);

- metoda pentru a scrie un element  $\leftrightarrow$    
 bec iterable  $\leftrightarrow$ . contains,  $\top$  iterable  $\leftrightarrow$ . firstWhere,  $\top$  iterable  $\leftrightarrow$ . elementAt,  $\top$  iterable  $\leftrightarrow$ . lastWhere;   
 $\hookrightarrow$  .firstWhere și .lastWhere prezintă ca  
parametru o funcție;

### Lists

- liste fixe și generabile;
- sunt templateate;
- liste tip referință;
- creare nouă liste:
  - $\hookrightarrow$  var l = [1, 2, 3]; // List<int> // from an
  - var l1 = List<int>.from([1, 2, 3]); // empty list  
// iterable
  - var l2 = <int>[]; // empty list  
// type must be specified (can't be inferred)

- proprietăți:
  - $\hookrightarrow$  var l = ["hello", "dant", "sss", "alergăting"];
   
present(l, first); // hello
   
present(l, last); // alergăting
   
present(l, isEmpty); // false
   
present(l, length); // 4
   
// returns an iterable
   
// (length, sss, dant, hello)
   
print(l, reversed);

- constructori marcat  $\leftrightarrow$  .empty, .filled, .generate, .of (la fel ca .from, deoarece se prezintă un iterabil de același tip ca lista, nu de același tip), .from, .unmodifiable (tot, în afară de .generate care acceptă parametru bec iterable .unmodifiable acceptă parametru bec generabile pentru a specifica dacă acesta adaugă la ea);

→ nice object who doesn't doin dynamics ( $\Rightarrow$  deet tique me grante je informed, we consideré dynamics),

↳ var l1 = [1, 2, 3];  
 print(l1, numType); // List<int>  
 var l2 = [ ]; // l2. numType = List<dynamics>  
 var l3 = List<empty(); // l3. numType = List<empty>  
 // l4. numType = List<empty>. empty();  
 var l4 = List<empty>. empty();

→ la. funt deed me dann een tip, me se face type inference en lista/ struktural dat ca parameter (lista resultaat van je List<dynamics>)  
 ja la. if se face;

↳ var l1 = List<int>. from([1, 2, 3]); // List<int>  
 var l2 = List<from([1, 2, 3]); // List<dynamics>  
 var l3 = List<int>. of([1, 2, 3]); // List<int>  
 var l4 = List<of([1, 2, 3]); // List<int>

→ constructorul . generate  $\Leftrightarrow$  regulaare unei liste după o relație matematică intre index și valoarea corespondătoare (parametrul ca parametru nr. de elemente din lista și o funcție care primește indexul și returnează valoarea);

↳ int get(int i) { return i%3; }  
 main() { // [0, 1, 4, 9, 16], i informed  
 var l = List<int>. generate(s, (i) => i+i);  
 // l1 = [0, 1, 2, 3, 4]  
 var l1 = List<int>. generate(s, (i) => i+i);  
 // l2 = [0, 1, 2, 0, 1]  
 var l2 = List<int>. generate(s, get); }

→ în List <dynamic> putem pune elemente de către user, me trebuie să aibă același tip tipitate;

```

    L var l = [10, "test", 1.5]; // List < dynamic
    point(l[0], uint8Type); // uint
    point(l[1], uint8Type); // String
    var l1 = <int>[10, "test", 1.5]; // error
    var l2 = [1, 2, 3]; // List < int>

```

→ operator + =  $\Leftrightarrow$  crează un nou obiect, nu se adauge la lista existentă ( $T+V \Leftrightarrow T=T+V$ );

→ metoda .add și .addAll adaugă elemente la obiectul curent;

```

    L var l = [1, 2, 3]; // addAll  $\Leftrightarrow$  add an iterable of T
    l.add(4); // l = [1, 2, 3, 4], add one element
    l.addAll([10, 20, 30]); // l = [1, 2, 3, 4, 10, 20, 30]
    var l1 = [1, 2, 3];
    l1 += [4]; // l1 = [1, 2, 3, 4]
    l1 += [10, 20, 30]; // l1 = [1, 2, 3, 4, 10, 20, 30]
    var l2 = List < int > .from([1, 2, 3], // fixed list
    growable: false); // growable: false
    l2.add(5); // error

```

→ deosebit operator + = crează un nou obiect care este deosebit, adică curent, chiar dacă să fieță este fixed, pentru că + = și să devină growable;

```

    L var l = List < int > .from([1, 2, 3], // fixed list
    growable: false); // growable list
    l += [4]; // l = l + [4] = [1, 2, 3, 4], growable list
    l.addAll([5]); // l = [1, 2, 3, 4, 5]

```

→ constructorul .filled  $\Leftrightarrow$  populează o listă cu un singur valoare (permite ca parametrii să de-

éléments de la liste, valeur);  
↳ van  $l = \text{List}\langle \text{int} \rangle$ . filled(5, 0, générable: false),  
point(2); // [0, 0, 0, 0, 0], filled list  
 $l[0] = 1$ ;  $l[1] = 2$ ;  $l[3] = 3$ ; //  $l = [1, 2, 0, 3, 0]$   
 $l[10] = 100$ ; // error, index out of range

→ ex. filled se fait sur une matrice;  
↳ //  $l = [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]$   
van  $l = \text{List}\langle \text{List}\langle \text{int} \rangle \rangle$ . filled(  
5, 3, // rows  
3, // columns  
0, // value (dégout)  
générable: false),  
générable: false);

→ si liste se fait avec un opérateur de cascade..  
↳ méthode .addAll (générale de la participation à  
type `Sequence`);  
↳ van  $l = \langle \text{int} \rangle [ ] .. \text{addAll}([1, 2, 3])$ ; // List<int>  
van  $l = [ ] .. \text{addAll}([1, 2, 3])$ ; // List<dynamics>

→ méthode .insert, .insertAll ajoutent un élément  
à la fin ou à l'index indiqué, il faut donner un élément,  
un caractère de séparateur et l'index de l'élément.  
↳ van  $l = ["A", "B", "C"]$ , "D", "E", "F";  
l. insert(2, "XX"); // l = [A, B, XX, C, D]  
l. insertAll(0, ["a", "w", "e"]); // l = [a, w, e, A, B, C]  
// insert at the end => index = l.length  
// l = [a, w, e, A, B, XX, END];  
l. insert(l.length, "END");  
l. insert(100, "XX"); // error => inserted index

- metode . remove , . removeAt , . removeRange ,
  - . removeLeft() , . removeLast() gheeft den laatste een element , enkel de la een index , enkel dentro van range , enkel dupé o gracie , uitsluitend element den leste ;
- metode . retainWhere pastreekt / selecteert elementen van range o enkeltyke gracie si
  - elementen van range o enkeltyke gracie si
  - metode . clear() gheeft hele elementen den leste ,
  - de la start de sond - 1;
- la . removeRange  $\leftrightarrow$  de la start de sond - 1 ;
  - ↳ van  $l = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$  ;
    - $l = [1, 2, 3, 4, 5, 6, 7, 8, 9]$
    - print( $l$ . removeLast()); // 10
    - $l = [1, 2, 4, 5, 6, 7, 8, 9]$
    - print( $l$ . remove(2)); // 13
    - $l = [1, 5, 6, 7, 8, 9]$
    - print( $l$ . removeRange(1, 3)); // returns used
    - $l = [1, 5, 7, 8]$
    - l . removeWhere((x) => x % 2 == 0) ;
      - $l = [1, 5, 7, 9]$
      - l . remove(1000) ; // false  $\leftrightarrow$  not found
    - print( $l$ . remove(1)) ; // true  $\leftrightarrow$  found
    - $l = [5, 7, 9]$
    - print( $l$ . remove(1)) ; // true  $\leftrightarrow$  found

- metode . contains , . indexOf , . lastIndexOf ,
  - . indexOfWhere seten true / false
    - (. contains) , . indexWhere seten true / false de gracie van element den leste , la een enkeltyke gracie dupé o gracie (. indexWhere is specifiek index , dupé o gracie gracie ) ;
  - index van  $l = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$  ;
    - print( $l$ . indexOf("1000")); // -1  $\leftrightarrow$  not found
    - print( $l$ . indexOf("4")); // 3  $\leftrightarrow$  found
    - print( $l$ . indexOf("2")); // 1  $\leftrightarrow$  found
    - print( $l$ . indexOfWhere((x) => x % 2 == 0)); // 1 , l[1] = 2

```

    " " g . e [ s ] = 10
    print( l . lastIndexWhere ( ( i ) => i . length == 0 ) );
    print( l . contains ( s ) ); // true
    print( l . contains ( 1000 ) ); // false

    // firstWhere , . lastWhere from iterable
    var d1 = [ "Dent" , "Hello" , "World" ];
    var e = d1 . firstWhere ( ( i ) => i . length == 4 );
    print ( e ); // Dent
    print ( d1 . firstWhere ( ( i ) => i . length == 10 ) );
    var g = d1 . firstWhere ( () => "not found" );
    print ( g ); // null : () => "not found"

    var g = d1 . lastWhere ( ( i ) => i . length == 5 );
    print ( g ); // not found
    print ( d1 . lastWhere ( () => "not found" ) );
    print ( g ); // null : () => "not found"

    var j = d1 . firstWhere ( ( i ) => i . length == 100 );
    print ( j ); // null : () => "not found"
    print ( d1 . firstWhere ( () => "not found" ) );
    print ( j ); // null : () => "not found"

    // enter ( no argument )
    " " enter

```

→ metodele .getRange , .subList , .reduce returnează  
un iterable , o listă , un element de tip T dintr-o  
listă de tip T și o listă de tip T rezultată dintr-un  
range , de la un start (optional end) , cel  
care se folosește și funcție (pe lângă cele din  
iterable .skip , .count etc.) ;  
 $\ell = [1, 2, 3, 4, 5]$ ;  $\langle \text{list} \rangle((i) \Rightarrow i * i)$ ;  $\|l_1 = (1, 4, 9, 16, 25)$

van  $l = [1, 2, 3, 4, 5]$  ;  $l[1] = (1, 4, 9, 16, 25)$   
 van  $l_1 = l.map(\text{int})$  (map)  $((i) \Rightarrow i + 1)$ ;  $l[1] = (1, 4, 9, 16, 25)$   
 van  $l_2 = l_1.map(\text{int})$ , skip geest (count) elements  
 $l[1] = (5, 16, 25)$ , skip(2);  
 van  $l_3 = l_2.map(\text{int})$  ;  
 $= (6)$   
 van  $l_4 = l_3.where((i) \Rightarrow i \% 2 == 0)$  ;  
 $l[1] = (9, 25)$   
 $l[1] = l.map(\text{int})$  (map)  $((i) \Rightarrow i + 1)$ . skip(2)  
 $l[1] = l.map(\text{int})$  (map)  $((i) \Rightarrow i \% 2 != 0)$  ;  
 $.where((i) \Rightarrow i \% 2 != 0)$  ;

$\| [9, 25] = 15$   
van  $l5 = 14. totlast();$   
 $\| forEach reduce$   
van  $l6 = [1, 2, 3, 4, 5];$

van  $s = 0;$   
 $\| s + i; \| ss = 15$

l.forEach((i) => s += i);

van  $l7 = [1, 2, 3, 4, 5, 6];$   
 $\| point(l, reduce((sum, i) => sum + i)); \| 21$

point(l, reduce((sum, i) => sum + i));

→ methode . sort sorteert elementen van lijst  
(enkele defunct, doce me i se de voren  
parametere si i se grote da o functie co  
parametere wie se especie module / criterium de  
sortare);

↳ van  $l = ["Hello", "Dant", "Tom", "Alaythya"];$   
 $\| l.sort(); \| l = [Alaythya, Dant, Hello, Tom]$   
 $\| l = [Tom, Dant, Hello, Alaythya];$   
 $\| l.sort((i, j) => i.length - j.length);$

→ argumentenlii eenie de commando sunt sorte (parametere  
gundie main  $\leftrightarrow$  List<String>);

### Sets

→ se gesorteerde 2...1;  
→ un etenabili;  
→ time door nr. unice;  
→ creare unie set;

↳ van  $s = \{1, 1, 5, 6, 7, 5, 3\};$   
 $\| s = \{1, 5, 6, 7, 3\}$

van  $s1 = \{\} ; \| empty set with type$   
 $\| Set<int>; \| empty set$

van  $s2 = Set<int>(); \| empty set$

- constructori named  $\Rightarrow$  . identity(), . of , . from , . unmodifiable (unctionele 3 sunt cu acelasi logica ca la lista),  
 ↳ var s1 = Set<int>.identity(); // s1 = <int>{1, 2, 3}  
 ↳ var s2 = <int>{1, 2, 3};  
 ↳ var s3 = Set<int>.from([1, 2, 3, 1, 2, 3]);
- metodele . add , . addAll adaugă un element , un storable de elemente în set ;
- var s = {1, 2, 3};  
 ↳ s.add(4); // s = {1, 2, 3, 4}  
 ↳ s.addAll([1, 3, 5]); // s = {1, 2, 3, 4, 5}.
- operatorul + = nu poate fi folosit (operatorul + nu este suprascrierat);
- metodele . remove , . removeAll , . removeWhere , . retainWhere , . retainAll , . clear sterge elemente din set (aceazi logica ca la lista , la . retainAll se păstrează toate din storablele date ca parametru);  
 ↳ var s = {1, 2, 3, 4, 5, 6, 7, 8};  
 ↳ s.removeWhere((x) => x % 2 == 0);  
 ↳ s = {3, 5, 7};  
 ↳ s.removeAll([1, 3, 9, 10, 11]);
- metodele . intersection , . difference , . union returnă o nouă set obținut prin intersecție , diferență , sau unirea celor două cu storable parametrul;

```

    ↳ var s1 = {1,2,3,4}, s2 = {3,4,5,6};
      print(s1.intersection(s2)); // {3,4}
      print(s1.union(s2)); // {1,2,3,4,5,6}
      print(s1.difference(s2)); // {1,2}
  
```

→ methods: .lookup, .containsAll, .contains  
 returniert element / null, true / false, true/false  
 für ganzes Objekt ob es ein Element, un storable,  
 ein Element ob gesucht im set;

```

    ↳ var s = {1,2,3,4};
      print(s.contains(2)); // & true
      print(s.contains(200)); // false
      print(s.lookup(2)); // 2
      print(s.lookup(200)); // null
      print(s.containsAll({1,2,3})); // true
      print(s.containsAll([1,2,1,2])); // true
      // set <- {1,2} => true
      print(s.containsAll({1,2,3,4,5})); // false
      print(s.containsAll([1,2,3,4,5])); // false
  
```

### Maps

→ die gesuchte  $\downarrow \langle \text{key} \rangle : \langle \text{value} \rangle \uparrow$ ,  
 → un storable;  
 → createe eine Map:

```

    ↳ var m = {"Peppe": 10, "Janne": 9, "George": 7};
      var o = {String, int} > int; // empty map
      var s1 = Map<String, int>(); // empty map
  
```

→ constructor named  $\leftrightarrow$  .identity(), .of, .from,  
 .unmodifiable (ca la Set, da er parametrische Map);

L ↳ var m1 = Map<String, int>.identity();  
 || m1 = <String, int>[]  
 || m2 = [1:1, 2:4]  
 var m2 = Map<int, int>.from([1:1, 2:4]);

→ MapEntry<K, V> (K key, V value)  $\leftrightarrow$  a tuple de  
 2 éléments (see propriété .key, .value);  
 → constructeur / méthode statique ( $\leftrightarrow$  .fromEntries),  
 .fromIterables, .fromIterable (meme ca  
 paramètre un iterable de valeur, un  
 iterable de K chez si une de valeur, un  
 iterable de éléments si optionnel a fonction  
 iterable de element dynamic et il transforme  
 première un element dynamic et il transforme  
 autre  $\rightarrow$  chose si a fonction similaire pour ce  
 valeur);

→ its .fromIterables, ca 2 iterables paramètres  
 trouvés et cette accepte « éléments »  
 || m = Map<String, Ge>.of("Rö", 10, Ge::2, "Ge", 3)  
 var m = Map<String, MapEntry<String, int>>.  
 MapEntry("Rö", 10, Ge::2, "Ge", 3),  
 MapEntry("Ge", 3));

|| m1 = Map<String, Ge>.of("Rö", 10, Ge::2, "Ge", 3);  
 var m1 = Map<String, Ge>.fromIterable([["Rö", "Ge", "Ge"],  
 [10, 8, 6]]);

var dict = [[["Rö", 10], {"Ge": 8}], [{"Ge": 6}],  
 || m2 = Map<String, Ge>.of("Rö", 10, Ge::8, "Ge", 6);  
 var m2 = Map<String, Ge>.fromIterable(dict,  
 key: (e) => e[0],  
 value: (e) => e[1]);

→ opération []  $\leftrightarrow$  map-name [key] = value;

→ metodele . addEntries , . addAll adds up la un map un iterable de MapEntry , un alt map , map un iterable

↳ var m = <String, int>[];  
m["Ro"] = 5; // m = {Ro: 5};  
m.addEntries(MapEntry("Ro", 10),  
m.addEntries(MapEntry("Ge", 7)));  
// m = {Ro: 5, Ro: 10, Ge: 7}  
// m = {Ro: 5, Ro: 10, Ge: 7, Te: 7}  
m.addAll({ "Te": 7});  
// m = {Ro: 5, Ro: 10, Ge: 7, Te: 7}  
m["Ge"] = 6;

→ metodele . putIfAbsent , . update , . updateAll  
deo update la elementele din map (unul , unde , unde) si returneaza parametru o cheie si o functie pentru care se va cheia deo update , o cheie si o functie de update (optional ) , si o functie care este cheie respectiva , si o functie de update ;

↳ var m = {"Ro": 10, Ge: 8, Tom: 5};  
// m = {Ro: 10, Ge: 8, Tom: 5};  
m.putIfAbsent("Tom", () => 5);  
// m = {Ro: 8, Ge: 8, Tom: 5};  
m.update("Ro", () => v - 2);  
// m = {Ro: 8, Ge: 8, Tom: 5, Tomel: 8};  
m.update("Tom", () => 10, () => Absent: () => 8);  
// m = {Ro: 4, Ge: 8, Tom: 1, Tomel: 4};  
m.updateAll((k, v) => v - 4);

→ metodele . remove , . removeAll , . clear sterg elemente si sunt similar cu cele din Set (nu multe

restauranteur băl si ultimul în "vad";

↳ variabilă m = { "Raul": 10, "Alina": 8, "Ion": 5,  
 "Gabi": 7};  
 || m = { Raul: 10, Alina: 8, Gabi: 7 }  
 m.remove("Ion"); Gabi: 7  
 || m = { Raul: 10, Gabi: 7 };  
 m.removeWhere((k,v) => k.length >= 5);

- .remove primește ca parametru o cheie și .removeWhere = specifică pe cheie și valori;
- metoda .containsKey , .containsValue verifică dacă o cheie, o valoare există în map (returnând true sau false);
- ↳ variabilă m = { "Alin": 4, "Maria": 8, "Dragos": 5 };  
 print(m.containsKey("Alin")); || true  
 print(m.containsValue(2)); || false
- puntem o stoc puntem-un map, se generează proprietate .entries, .keys, .values (returnând un iterable de entries (cheie, valoare), de cheie, de valoare);
- ↳ variabilă m = { "RE": 7, "Ma": 8, "Da": 5 };  
 print(m.values); || (7,8,5)  
 print(m.keys); || (RE, Ma, Da)  
 for (var i in m.entries)  
 || MapEntries( RE: 7 ), MapEntry(Ma: 8 )  
 || MapEntry(Da: 5)  
 print(i);

## Variable și Jurnal

- const și final;
- const ( $\Rightarrow$  valoare / expresie care nu poate fi schimbată în timpul compilării);
- final ( $\Rightarrow$  variabilă care nu se schimbă (constanță);
- final trebuie să fie initializată înainte de a fi utilizată;
- const trebuie să fie valoare care nu poate fi modificată (nu poate fi ceea ce nu poate fi evaluat de compilator), în timp ce pentru final poate fi orice valoare (nu și mai poate fi schimbată pe parcursul vieții variabilei);
- variabilele sunt și final și const;

↳ const int x = 10;

const y = 20;

final int x1 = 10;

final y = 20;

final const x = 10; // eror

- nici variabilele const, nici cele final nu mai pot fi schimbată;

↳ const int x = 10; final y = 20;

const int x = 10; // eror

y = 20; // eror

x1 = 10; // eror

final int x1 = Random().nextInt(100);

const int y1 = Random().nextInt(100); // eror

## Exceptions

- try { }  
catch (Exception) { }

→ try {  
 on <ExceptionType1> [catch(e)] }  
 on <ExceptionType2> [catch(e)] }  
 ---  
 catch(e) }  
 finally(e) }

↳ dynamic x = "Test";  
 try { x += 2;  
 catch(e) { print("Error"); }  
 try {  
 x += 2;  
 } catch(e) {  
 print("Error");  
 } finally { print("Done"); }  
 } Error, Done

→ ↳ not if uncaught exception or caught but threw.  
 ↳ void foo(Last x) { x > 10 }  
 main() { try { foo(20);  
 catch(e) { print("\$e"); } } } Error 20  
 ↳ for catch pattern same as stack-els

↳ try { foo(20);  
 catch(e,s) { print("\$e"); print(s); } } Error 20

→ pattern free sethrows (exception is of transient in  
 catch-all separator);  
 try { try { throw "some exception"; }  
 catch(e) { print("error = \$e"); } } sethrows;  
 try { print("second catch"); }  
 catch(e) { print("second exception, second catch"); }  
 "error = some exception"