Collections et Itérateurs

Yann Baës - Pierre Corbel - Amandine Watrelos

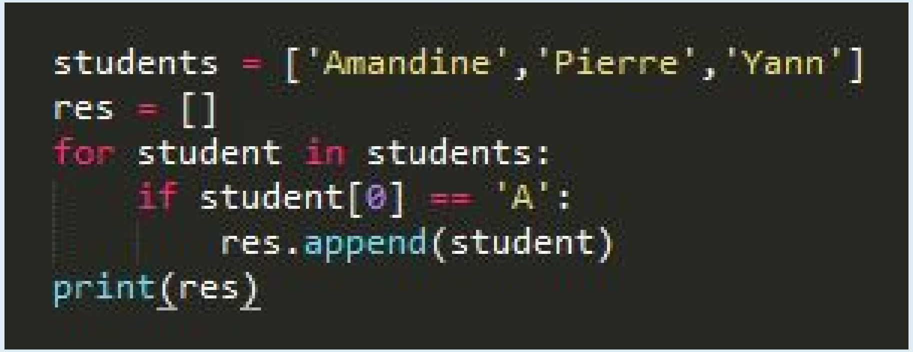
Université de Lille 1 SCIENCES ET TECHNOLOGIES

UFR IEEA Formations en Informatique de Lille 1

Plan

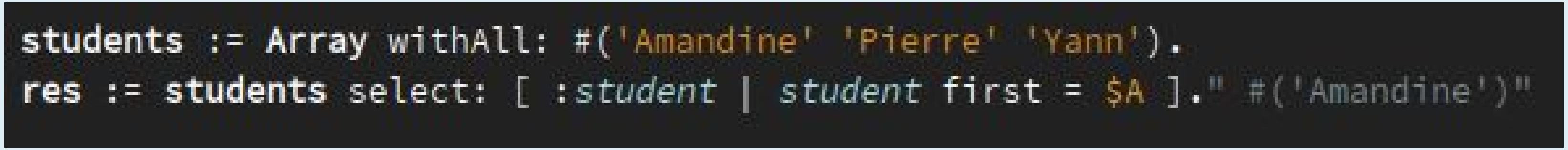
1) **Les collections**
   a) OrderedCollection
   b) Array
   c) Set
   d) Dictionary

2) **Les itérateurs**

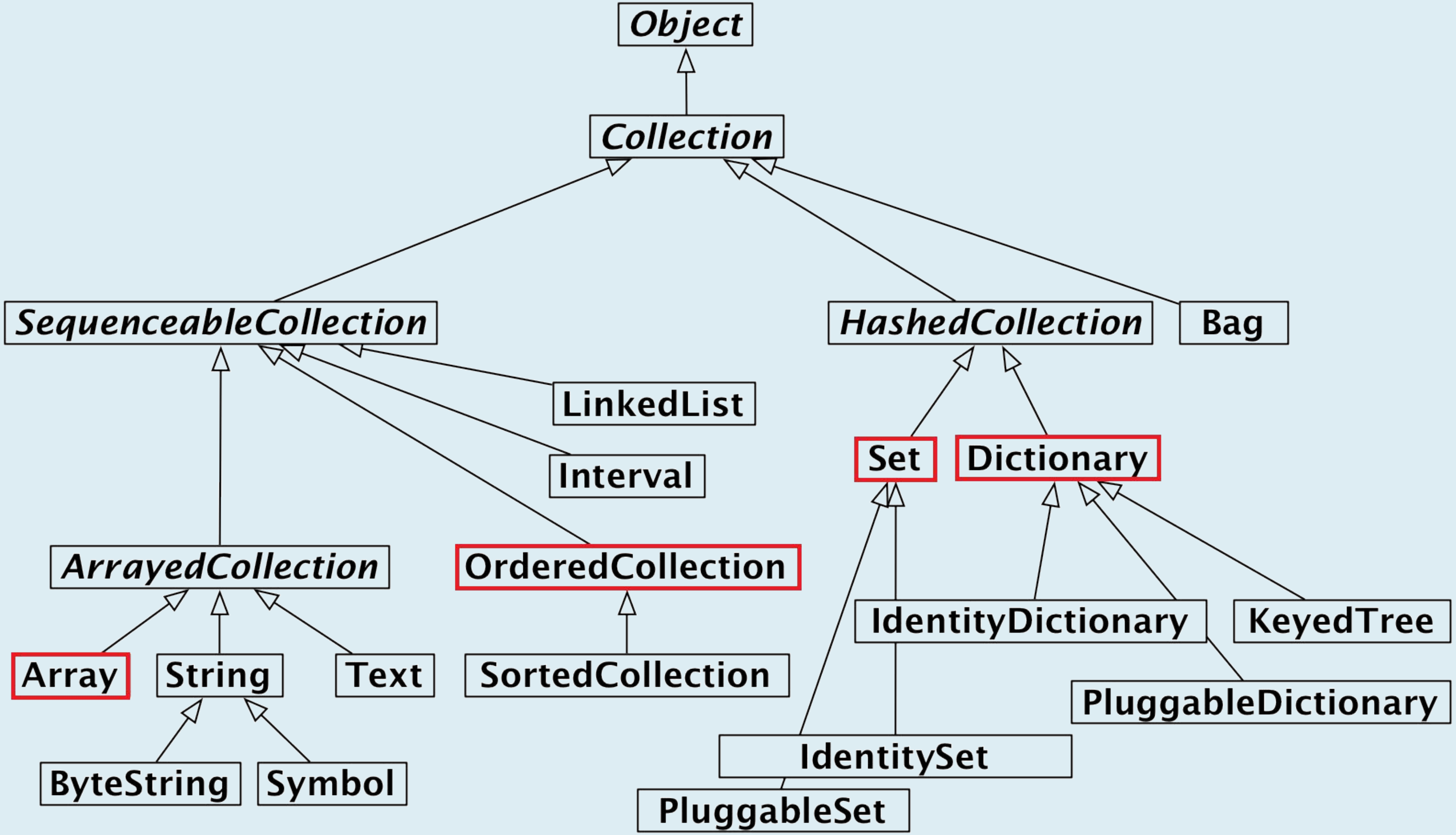


```python
students = ['Amandine','Pierre','Yann']
res = []
for student in students:
    if student[0] == 'A':
        res.append(student)
print(res)
```

VS



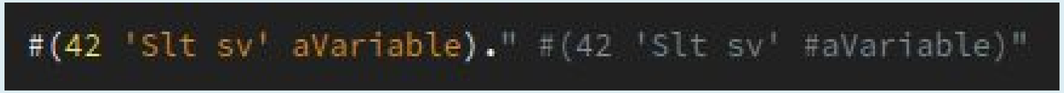
```smalltalk
students := Array withAll: #('Amandine' 'Pierre' 'Yann').
res := students select: [ :student | student first = $A ]." #('Amandine')"
```


Object
Collection
SequenceableCollection
HashedCollection
Bag
LinkedList
Interval
Set
Dictionary
ArrayedCollection
OrderedCollection
IdentityDictionary
KeyedTree
Array
String
Text
SortedCollection
PluggableDictionary
IdentitySet
ByteString
Symbol
PluggableSet

Collections

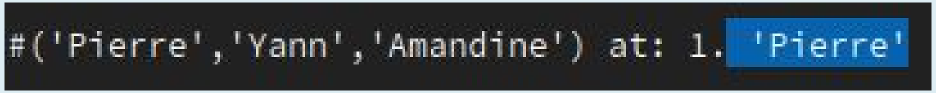
- Simples à utiliser

- Peut contenir n'importe quel objet



```
#(42 'Slt sv' aVariable)." #(42 'Slt sv' #aVariable)"
```

- Index du 1er élément à 1

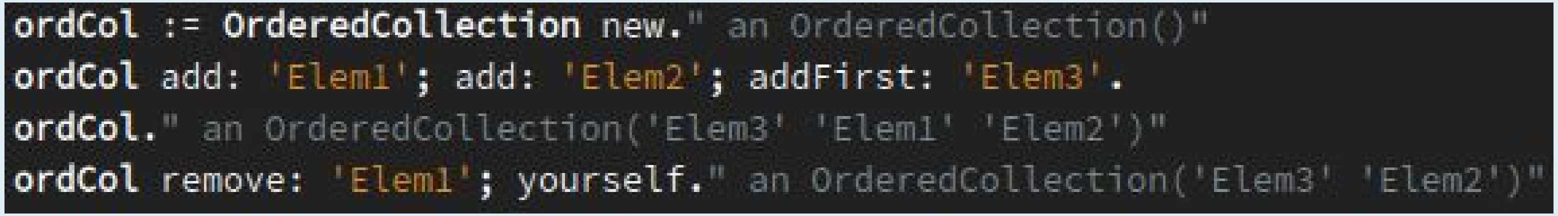


```
#('Pierre','Yann','Amandine') at: 1. 'Pierre'
```

Protocole	**Méthodes**
accessing	size, capacity, at: *anIndex*, at: *anIndex* put: *anElement*
testing	isEmpty, includes: *anElement*, contains: *aBlock*, occurrence-sOf: *anElement*
adding	add: *anElement*, addAll: *aCollection*
removing	remove: *anElement*, remove: *anElement* ifAbsent: *aBlock*, removeAll: *aCollection*
enumerating	do: *aBlock*, collect: *aBlock*, select: *aBlock*, reject: *aBlock*, detect: *aBlock*, detect: *aBlock* ifNone: aNoneBlock, inject: *aValue* into: *aBinaryBlock*
converting	asBag, asSet, asOrderedCollection, asSortedCollection, asArray, asSortedCollection: *aBlock*
creation	with: *anElement*, with:with:, with:with:with:, with:with:with:with:, withAll: *aCollection*

OrderedCollection

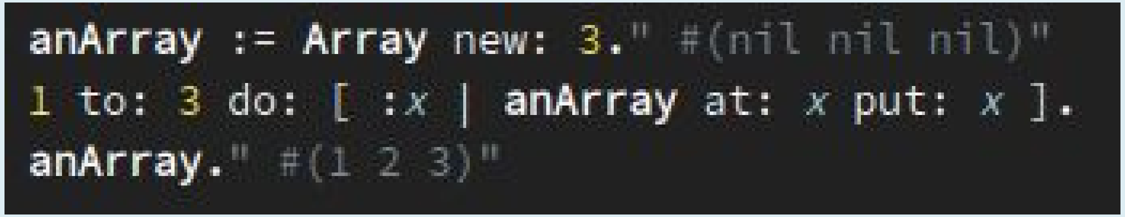
- Taille dynamique (ajout d’éléments)



```
ordCol := OrderedCollection new." an OrderedCollection()"
ordCol add: 'Elem1'; add: 'Elem2'; addFirst: 'Elem3'.
ordCol." an OrderedCollection('Elem3' 'Elem1' 'Elem2')"
ordCol remove: 'Elem1'; yourself." an OrderedCollection('Elem3' 'Elem2')"
```

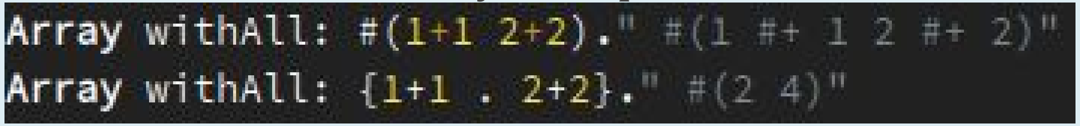
Array

- Taille fixe



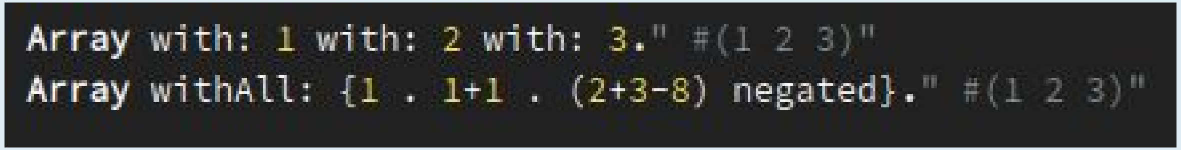
```
anArray := Array new: 3." #(nil nil nil)"
1 to: 3 do: [ :x | anArray at: x put: x ].
anArray." #(1 2 3)"
```

- Création littérale vs dynamique



```
Array withAll: #(1+1 2+2)." #(1 #+ 1 2 #+ 2)"
Array withAll: {1+1 . 2+2}." #(2 4)"
```

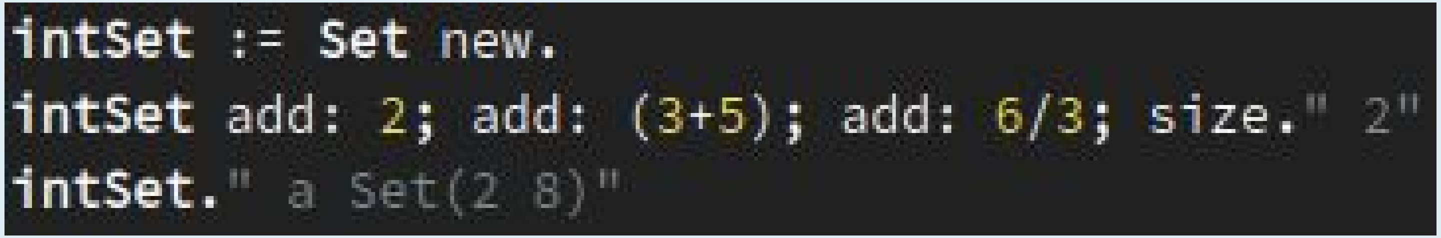
- Différentes méthodes de création :



```
Array with: 1 with: 2 with: 3." #(1 2 3)"
Array withAll: {1 . 1+1 . (2+3-8) negated}." #(1 2 3)"
```

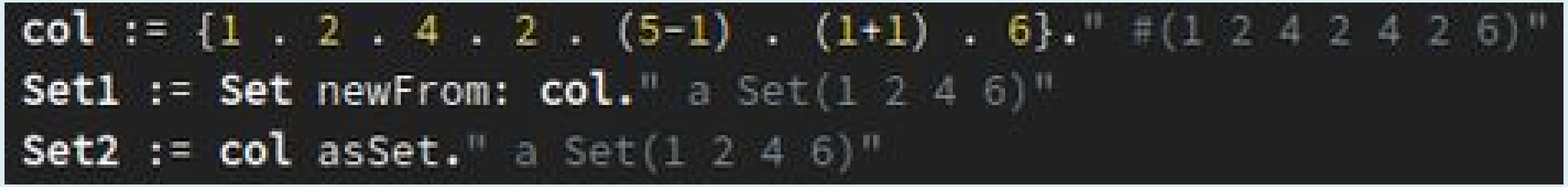
Set

- Aucun doublon possible



```
intSet := Set new.
intSet add: 2; add: (3+5); add: 6/3; size." 2"
intSet." a Set(2 8)"
```

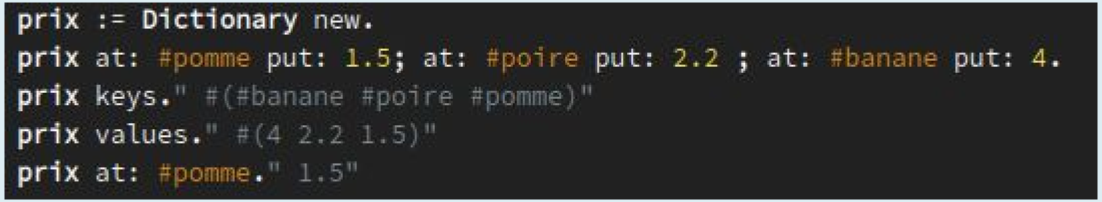
- Différentes méthodes de création



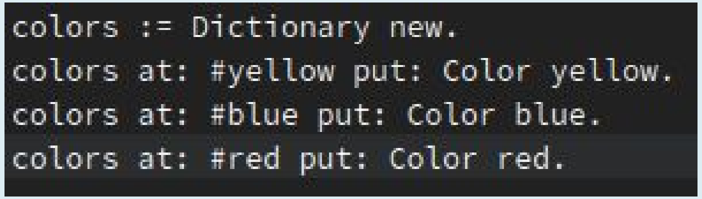
```
col := {1 . 2 . 4 . 2 . (5-1) . (1+1) . 6}." #(1 2 4 2 4 2 6)"
Set1 := Set newFrom: col." a Set(1 2 4 6)"
Set2 := col asSet." a Set(1 2 4 6)"
```

Dictionary

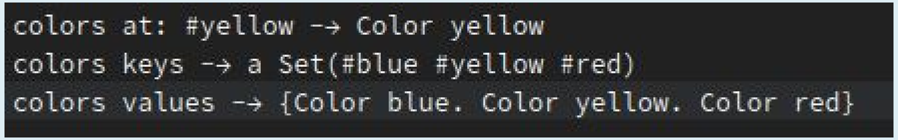
- A chaque clef est associée une valeur



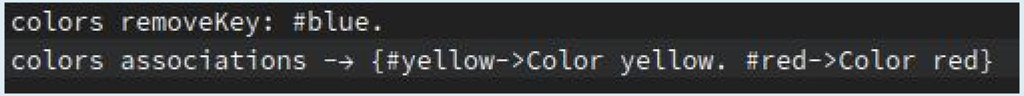
```
prix := Dictionary new.
prix at: #pomme put: 1.5; at: #poire put: 2.2 ; at: #banane put: 4.
prix keys." #(#banane #poire #pomme)"
prix values." #(4 2.2 1.5)"
prix at: #pomme." 1.5"
```



```
colors := Dictionary new.
colors at: #yellow put: Color yellow.
colors at: #blue put: Color blue.
colors at: #red put: Color red.
```



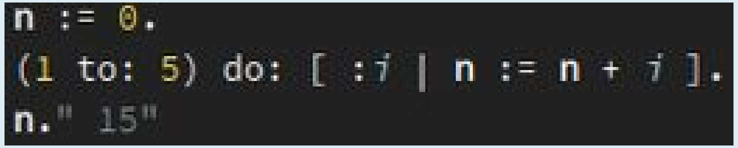
```
colors at: #yellow -→ Color yellow
colors keys -→ a Set(#blue #yellow #red)
colors values -→ {Color blue. Color yellow. Color red}
```



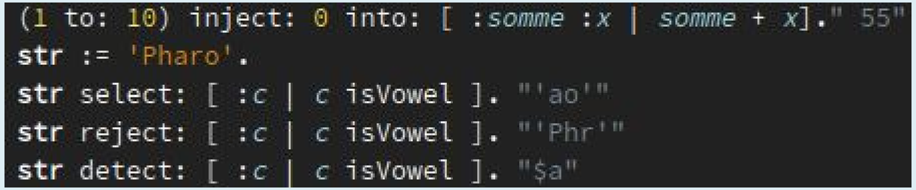
```
colors removeKey: #blue.
colors associations -→ {#yellow->Color yellow. #red->Color red}
```

Iterator

- Envoi de messages à des blocs, entiers ou collections



```
n := 0.
(1 to: 5) do: [ :i | n := n + i ].
n." 15"
```



```
(1 to: 10) inject: 0 into: [ :somme :x | somme + x]." 55"
str := 'Pharo'.
str select: [ :c | c isVowel ]. "'ao'"
str reject: [ :c | c isVowel ]. "'Phr'"
str detect: [ :c | c isVowel ]. "$a"
```