Colecciones:

- **OrderedCollection**
  - Es una colección ordenada y mutable donde los elementos mantienen el orden en que se agregan. Permite duplicados y puede crecer o reducirse dinámicamente.
  - oc := OrderedCollection new.
  - oc add: 5.
  - oc add: 3.
  - oc add: 5.
  - ^ oc "=> #(5 3 5)"
- **SortedCollection**
  - Es una colección ordenada automáticamente según un criterio de comparación (por defecto <), siempre ordenada internamente. También mutable.
  - | sc |
  - sc := SortedCollection new.
  - sc add: 5.
  - sc add: 3.
  - sc add: 5.
  - ^ sc "=> #(3 5 5)"
- **Bag**
  - Es una colección no ordenada que permite duplicados y mantiene el conteo de cuántas veces aparece cada elemento (multiconjunto).
  - | bag |
  - bag := Bag new.
  - bag add: 5.
  - bag add: 3.
  - bag add: 5.
  - bag occurrencesOf: 5. "=> 2"
- **Dictionary**
  - Colección de pares clave-valor (mapa/hash). Las claves son únicas y se accede a los valores mediante las claves.
  - | dict |
  - dict := Dictionary new.
  - dict at: 'a' put: 1.
  - dict at: 'b' put: 2.
  - dict at: 'a'. "=> 1"
- **Array**
  - Colección de tamaño fijo una vez creada. Es indexada y ordenada. Puede contener duplicados y cualquier tipo de objeto.
  - | arr |
  - arr := Array with: 1 with: 2 with: 3.
  - arr at: 2. "=> 2"
- **Set**
  - Colección de elementos únicos sin orden definido. No permite duplicados.
  - | set |
  - set := Set new.

    - set add: 1.
    - set add: 2.
    - set add: 1.
    - set size. "=> 2"
- **Matrix**
    - Colección multidimensional rectangular, comúnmente una matriz matemática. En Smalltalk es una subclase de Array que representa tablas bidimensionales.
    - | m |
    - m := Matrix rows: 2 columns: 2.
    - m at: 1 at: 2 put: 5.
    - m at: 1 at: 2. "=> 5"

Métodos:

- **#collect**
    - Transforma cada elemento de una colección aplicando una función, y devuelve una nueva colección con los resultados.
    - #(1 2 3) collect: [:n | n * 2] "=> #(2 4 6)"
- **#select**
    - Filtra los elementos que cumplen con una condición (los que hacen que el bloque devuelva true).
    - #(1 2 3 4) select: [:n | n even] "=> #(2 4)"
- **#inject: into:**
    - Es un acumulador (también conocido como fold o reduce en otros lenguajes). Comienza con un valor inicial y lo va combinando con los elementos.
    - #(1 2 3 4) inject: 0 into: [:sum :n | sum + n] "=> 10"
- **#do:**
    - Recorre la colección y ejecuta un bloque por cada elemento, pero no devuelve una nueva colección, solo ejecuta acciones.
    - #(1 2 3) do: [:n | Transcript show: n; cr]. "=> imprime 1 2 3 en consola"
- **#Reject:**
    - Recorre la colección y ejecuta un bloque por cada elemento, devolviendo una nueva colección con los elementos para los que el bloque devuelve false (los que no cumplen la condición).
    - #(1 2 3 4 5) reject: [:n | n even]. "=> #(1 3 5)"
- **#detect:**
    - Recorre la colección y ejecuta un bloque por cada elemento, devolviendo el primer elemento que hace que el bloque devuelva true.
    - #(2 4 5 6) detect: [:n | n odd]. "=> 5".
- **#detect:ifNone:**
    - idem a detect pero si ningún elemento cumple la condición, devuelve el valor del bloque pasado en ifNone: en lugar de lanzar un error.
    - #(2 4 6) detect: [:n | n odd] ifNone: [0]. "=> 0".
- **#keysAndValuesDo:**
    - Permite recorrer un diccionario (o colección asociativa), ejecutando un bloque que recibe dos argumentos: la clave y el valor de cada asociación.
    - Específico para colecciones que manejan pares clave-valor (por ejemplo, Dictionary).
    - d := Dictionary new.

    - d at: #a put: 1; at: #b put: 2.
    - d keysAndValuesDo: [:key :value | Transcript show: key; show: ': '; show: value; cr].
    - "=> a: 1, b: 2"
- **#keysDo:**
    - Recorre las claves de un diccionario (o colección asociativa) y ejecuta un bloque por cada clave.
    - d := Dictionary new.
    - d at: #a put: 1; at: #b put: 2.
    - d keysDo: [:key | Transcript show: key; cr]. "=> a. b"
- **#valuesAndCountsDo:**
    - Recorre una colección tipo Bag y ejecuta un bloque que recibe dos argumentos: el valor y su cantidad (count) en la bolsa.
    - Específico para colecciones que cuentan ocurrencias de elementos (multiconjuntos).
    - b := Bag new.
    - b add: 'a'; add: 'a'; add: 'b'.
    - b valuesAndCountsDo: [:v :c | Transcript show: v; show: ': '; show: c; cr].
    - "=> a: 2. b: 1.
- **#withAll:**
    - Crea una nueva colección agregando todos los elementos de otra colección al receptor, sin modificar el receptor original.
    - El receptor y el argumento deben ser colecciones.
    - c := OrderedCollection with: 1 with: 2.
    - c2 := c withAll: #(3 4). "=> c2 es una nueva colección con los elementos [1, 2, 3, 4]."
- **#includes:**
    - Pregunta si la colección contiene un elemento específico (devuelve true o false).
    - #(1 2 3) includes: 2. "=> true"
    - #(1 2 3) includes: 5. "=> false"
- **#includesKey:**
    - Pregunta si un diccionario contiene una clave específica (devuelve true o false).
    - d := Dictionary new.
    - d at: #a put: 1.
    - d includesKey: #a. "=> true"
    - d includesKey: #b. "=> false"