Guía de lenguajes

1. Principales funcionalidades comunes

	Smalltalk	Haskell	Prolog
Valores			
Comentarios	**"**un comentario**"**	**--**un comentario {- un comentario -}	%un comentario /*un comentario*/
Strings	**'**uNa CadEna**'**	**"**uNa CadEna**"**	**"**uNa CadEna**"**
Caracteres	**$**a	'a'	**97** (en un string)
Simbolos	#unSimbolo		unSimbolo 'unSimbolo'
Booleanos	**true false**	**True False**	
Array literal	**{**1. 2. 3**}** **{**1. $a. 'hola'**}**	**[**1, 2, 3**]**	**[**1, 2, 3**]** **[**1, a, "hola"**]**
Tuplas/functores		**(**1, 'a', "hola"**)**	**nombreFunctor(**1,a,"hola"**)**
Operadores lógicos			
Igualdad	**=** (igual) **==** (idéntico)	**==**	**=**
Diferencia	**~=** (no igual) **~~** (no idéntico)	**/=**	**\=**
Comparación	**> >= < <=**	**> >= < <=**	**> >= < =<**
"O" lógico	**|**	**||**	**;** **(1)**
"Y" lógico	**&**	**&&**	**,**
Negación	*unBool* **not**	**not** *unBool*	**not(** *Consulta* **)**
Operadores matemáticos			Solo a la derecha del **is**
Operaciones básicas	**+ - * /**	**+ - * /**	**+ - * /**
División entera	*dividendo* **//** *divisor*	**div** *dividendo divisor*	*dividendo* **//** *divisor*
Resto	*dividendo* **** *divisor*	**mod** *dividendo divisor*	*dividendo* **mod** *divisor*
Valor absoluto	*unNro* **abs**	**abs** *unNro*	**abs(***Nro***)**
Exponenciación	*base* **raisedTo:** *exponente*	*base* **^** *exponente*	*base* ****** *exponente*
Raíz cuadrada	*unNro* **sqrt**	**sqrt** *unNro*	**sqrt(***Nro***)**
Máximo ó mínimo entre dos números	*unNro* **max:** *otroNro* *unNro* **min:** *otroNro*	**max** *unNro otroNro* **min** *unNro otroNro*	
Valor ascii de un caracter	*unChar* **asciiValue**		
Caracter de un valor ascii	*unNro* **asCharacter**		
Colecciones y listas			
Longitud	*unaCol* **size**	**length** *unaLista*	**length**(Lista,Longitud)
Concatenación (sin efecto de lado)	*unaCol* **,** *unaCol*	*unaLista* **++** *otraLista*	**append**(Lista1,Lista2,Lista12)
Agregar un elemento	*unaCol* **add:** *unObjeto*	Las listas no se pueden modificar	
Patrón de listas		*unElem* **:** *unaLista*	**[** *Elem* **|** *Lista* **]**
Posición numérica	*unaCol* **at:** *unNro*	*unaLista* **!!** *unNro*	**nth1**(Nro,Lista,Elem)
Test de pertenencia	*unaCol* **includes:** *unObj*	**elem** *elemento lista*	**member**(Elem,Lista)
Máximo ó mínimo de un conjunto de números	*unaCol* **max** *unaCol* **min**	**maximum** *lista* **minimum** *lista*	
Sumatoria de un conjunto de números	*unaCol* **sum**	**sum** *lista*	**sumlist(***Lista,Sumatoria***)**

(1) **Importante** en algunos cursos se desalienta el uso del punto y coma, prefiriéndose que se separen las condiciones en varias cláusulas.

2. Smalltalk

A pesar de la variedad de mensajes la sintaxis del lenguaje siempre es **objeto mensaje**

Expresiones y métodos de uso común

| *unaVariable* **|**	Declara a *unaVariable* como variable local.
.	Indica separación entre sentencias.
self	Referencia al objeto receptor del mensaje que ejecutó el método.
nil	Objeto nulo.
unaVariable **:=** *unaExpresión*	Asigna *unaExpresion* a *unaVariable*
^*unObjeto*	Retorna *unObjeto* y termina la ejecución del método
[*sentencias* **]**	Constituye un bloque de código que contiene a las *sentencias*
UnaClase **new**	Crea y devuelve una nueva instancia de *unaClase*
unObjeto **isNil**	Devuelve true si el receptor es nil
unObjeto **notNil**	Devuelve true si el receptor no es nil
unObjeto **printString**	Devuelve un string que representa el contenido de *unObjeto*
unBool **ifTrue:** [*unasSentencias*] **ifFalse:** [*otrasSentencias*]	Ejecuta *unasSentencias* u *otrasSentencias* dependiendo del valor de verdad de *unBool.*
[*unBool*] **whileTrue:** [*sentencias*]	Ejecuta iterativamente las *sentencias* mientras *unBool* sea verdadero.
unNro **timesRepeat:** [*sentencias*]	Ejecuta iterativamente las *sentencias* exactamente *unNro* de veces.
unNro **to:** *otroNro* **do:** [:*indice* | *sentencias*]	Ejecuta iterativamente las *sentencias* la cantidad de veces comprendida entre *unNro* y *otroNro*. *Indice* varía en cada iteración, desde *unNro* hasta *otroNro*.

Clases de colecciones más comunes

- **Bag**: Tamaño variable, sin subíndice.
- **Set**: Tamaño variable, sin subíndice, no permite repetidos.
- **Array**: Tamaño fijo, con subíndice, orden de acuerdo al subíndice.
  En este sentido un String se comporta como un Array
- **OrderedCollection**: Tamaño variable, con subíndice, orden de acuerdo al subíndice.
- **SortedCollection**: Tamaño variable, con subíndice, orden de acuerdo a criterio que se especifica.
- **Dictionary**: Tamaño variable, acceso por clave, no permite claves repetidas

Los subíndices empiezan en 1.

Las colecciones con subíndice respetan el orden de los elementos en do: / select: / collect: / etc..

Para Dictionary do: / select: / collect: / etc. **funcionan sobre los valores** incluidos, no se tienen en cuenta las claves.

Métodos de colecciones

Para todas las colecciones	
unaCol **size**	Devuelve la cantidad de elementos que tiene *unaCol*
unaCol **includes:** *unObjeto*	Devuelve true si *unObjeto* se encuentra en *unaCol*.
unaCol **occurrencesOf:** *unObjeto*	Devuelve la cantidad de ocurrencias de *unObjeto* en *unaCol*.
unaCol **asBag** *unaCol* **asSet** *unaCol* **asOrderedCollection** *unaCol* **asArray**	Devuelve una **nueva** colección de la clase indicada con todos los elementos de *unaCol*.
unaCol **asSortedCollection:** [:*anterior* :*siguiente* | *unaCondicion*]	Devuelve una **nueva** colección con todos los elementos de *unaCol* ordenados según *unaCondicion*. *unaCondicion* es una expresión de valor booleano en la que intervienen *anterior* y *siguiente*. *anterior* quedará delante de *siguiente* cuando *unaCondicion* sea verdadera.
unaCol **do:** [:*unElem* | *sentencias*]	Ejecuta iterativamente las sentencias para cada *unElem* de *unaCol*. Retorna *unaCol*, en otras palabras no devuelve nada interesante
unaCol **select:** [:*unElem* | *unaExpr*]	Devuelve una **nueva** colección con los elementos de *unaCol* que hacen verdadero a *unaExpr*.*
unaCol **reject:** [:*unElem* | *unaExpr*]	Devuelve una **nueva** colección con todos los elementos de *unaCol* excepto los que hacen verdadero a *unaExpr*.*
unaCol **detect:** [:*unElem* | *unaExpr*] **ifNone:** *unBloque*	Devuelve el primer elemento de unaCol que hace verdadero a *unaExpr*. Si ninguno lo hiciera, se retornara la ejecución de *unBloque.**
unaCol **collect:** [:*unElem* | *sentencias*]	Devuelve una **nueva** colección con el resultado de evaluar iterativamente las *sentencias* para cada *unElem* de *unaCol.**
unaCol **inject:** *valorInicial* **into:** [:*acumulador* :*unElem* | *unaOperación*]	El *acumulador* empieza siendo el *valorInicial*. Luego se evalúa *unaOperación* para cada elemento en *unaCol*, y el resultado es puesto en el *acumulador*. Retorna el valor final del *acumulador.* (Ver foldl de Haskell) P.ej. para obtener la suma de los elementos de una colección *unaCol inject: 0 into: [:resul :elem | resul + elem]*
unaCol **allSatisfy:** [:*unElem* | *unaExpr*]	Devuelve true si todos los elementos de la colección hacen verdadera *unaExpr*.*
unaCol **anySatisfy:** [:*unElem* | *unaExpr*]	Devuelve true si algún elemento de la colección hace verdadera *unaExpr*.*
unaCol **union:** *otraCol*	Devuelve **una nueva colección** resultado de la unión de las dos anteriores, sin repetidos. Sin efecto sobre *unaCol* y *otraCol*.
unaCol **intersection:** *otraCol*	Idem *union:*, pero devuelve la intersección.
unaCol **count:** [:*unElem* | *unaExpr*]	Devuelve la cantidad de elementos de *unaCol* que hacen verdadera a *unaExpr*. Es equivalente a hacer *select:* con esa *unaExpr* y luego *size* del resultado.
unaCol **sum:** [:*unElem* | *unaSentencia*]	Devuelve la sumatoria de los valores que retorna *unaSentencia* por cada elemento de *unaCol*. Es equivalente a hacer *collect:* con esa *unaSentencia* y luego *sum* del resultado.
unaCol **detectMax:** [:*unElem* | *unaSentencia*]	Devuelven **el elemento de *unaCol*** que tiene mayor valor para *unaSentencia*. Nota: no devuelve el valor de *unaSentencia*, sino un elemento de *unaCol.*
unaCol **detectMin:** [:*unElem* | *unaSentencia*]	Idem *detectMax:*, pero devolviendo el elemento de menor valor.
unaCol **removeAllSuchThat:** [:*unElem* | *unaExpr*]	Remueve de *unaCol* los *unElem* que hagan verdadera a *unaExpr*. Devuelve **esa misma *unaCol* modificada**.
unaCol **gather:** [:*unElem* | *unaSentencia*]	Requiere que *unaSentencia* devuelva **una colección** para cada elemento de *unaCol*. Devuelve una **nueva colección** con todos los elementos de todas las colecciones que devolvió *unaSentencia*. Suele usarse para aplanar colecciones. Nota: En la implementación de Pharo, la colección que retorna es un Array de 100 elementos (medio feo).

* *unElem* referencia iterativamente a cada uno de los elementos de *unaCol*. *unaExpr* es una expresión de valor booleano en la que interviene *unElem*.

Sólo para colecciones de tamaño variable	
unaCol **add:** *unObjeto*	Agrega *unObjeto* a *unaCol*. **Devuelve *unObjeto***. Para las colecciones con subíndice se agrega al final.
unaCol **addAll:** *otraCol*	Agrega todos los elementos de *otraCol* a *unaCol.* Para las colecciones con subíndice se agregan al final. **Devuelve *otraCol***
unaCol **remove:** *unObjeto*	Elimina *unObjeto* de *unaCol.* **Devuelve *unObjeto.***
unaCol **removeAll**	Elimina todos los elementos de *unaCol*. **Devuelve *unaCol.***
Sólo para colecciones con subíndice (OrderedCollection,SortedCollection,Array,String)	
unaCol **,** *otraCol*	Devuelve una **nueva** colección con la concatenación de *unaCol* y *otraCol*. Respeta el orden.
unaCol **at:** *unNro*	Devuelve *el elemento* en la posición *unNro.*
unaCol **at:** *unNro* **put:** *unObjeto*	Coloca *unObjeto* en la posicion *unNro* de *unaCol*. Inválido para SortedCollection.
unaCol **first**	Devuelve el primer elemento de *unaCol* También hay **second** y otros.
unaCol **last**	Devuelve el último elemento de *unaCol*
unaCol **indexOf:** *unObjeto*	Devuelve la posición en la que aparece *unObjeto* dentro de *unaCol* (la primera si estuviera repetido); 0 si *unObjeto* no está en *unaCol*.
unaCol **beginsWith:** *otraCol*	*true* si … *unaCol* empieza con *otraCol*. También hay **endsWith:**
unaCol **copyFrom:** *unNro* **to:** *otroNro*	Devuelve una **nueva** colección con los elementos de *unaCol* comprendidos entre las posiciones *unNro* y *otroNro*.
unaCol **allButLast**	Devuelve una **nueva** colección con todos los elementos de unaCol excepto el último. También hay **allButFirst**.
UnaClase **new:** *unNro*	Devuelve una **nueva** colección de *UnaClase* de tamaño *unNro*
Sólo para Dictionary	
unaCol **at:** *unaClave*	Devuelve el valor asociado a *unaClave*, *nil* si unaClave no tiene asociado ningún valor.
unaCol **at:** *unaClave* **put:** *unObjeto*	Coloca *unObjeto* como valor asociado a *unaClave*.

3. Prolog

Un pequeño recordatorio de los predicados que usamos y qué relacionan o cuándo se verifican … para entenderlos bien, remitirse a lo que se vio en clase.
Ver algunos en la 1er página, en particular en la parte de colecciones y listas.
En los predicados de más de un argumento, la explicación respeta el orden de los argumentos.

not/1	recibe una consulta por parámetro, se verifica si el parámetro no da resultados. Ojo con la inversibilidad al usarlo.
findall/3	con un ejemplo: findall(S,tio(herbert,S),Sobr) liga Sobr con la lista de los S que verifican tio(herbert,S), en el orden en que el motor los va encontrando (que no se sabe cuál es). Entonces es findall(Que, Consulta, Lista). y liga Lista con los Que que satisfacen la consulta. Ojo con la inversibilidad al usarlo.
forall/2	con un ejemplo: forall(tio(herbert,S),amigo(S,milhouse)) se satisface si todos los S que verifican tio(herbert,S) verifican también tio(S,milhouse) Entonces es forall(Consulta1, Consulta2) y se verifica si todos las respuestas a Consulta1 son respuestas de Consulta2. Ojo con la inversibilidad y cómo trata a las variables

4. Haskell

Algunas funciones que se usan seguido. Ver algunas en la 1er página, en particular en la parte de colecciones y listas. Más info en http://haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html.
Recordar que los String son exactamente listas de chars, por lo tanto, todas las funciones que esperan una lista andan con un String. Las listas de String son entonces listas de listas de chars. P.ej:
filter isAlpha "pepe1234juan" (1) map head ["porque", "damos", "pan"]

head/1, tail/1	devuelven la cabeza y la cola de una lista
null/1	recibe una lista, devuelve true si es vacía, false si tiene al menos un elemento
map/2	recibe una función y una lista, y devuelve la lista resultante de aplicar la función a cada elemento de la lista. Onda el collect: de Smalltalk. P.ej.: map (2*) [1..5]
filter/2	recibe una condición (= función que devuelve booleano) y una lista, y devuelve la sublista de los que cumplen la condición. Onda el select: de Smalltalk. P.ej.: filter even [1..5]
all/2	recibe condición y lista, devuelve true si todos los elementos de la colección cumplen la condición, false en caso contrario. Onda allSatisfy: de Smalltalk.
any/2	recibe condición y lista, devuelve true si al menos un elemento de la colección cumple la condición, false en caso contrario. Onda anySatisfy: de Smalltalk.
foldl/3	Recibe una función, una semilla, y una lista. Toma la semilla y el primer elemento de la lista, los opera con la función, y el resultado se lo pasa nuevamente a foldl, como nueva semilla. Al final, devuelve ese valor "acumulado", resultado de operar recursivamente de a dos elementos: lo acumulado a cada paso, con el valor siguiente de la lista. (Ver inject:into: de Smalltalk) P.ej. para obtener la suma de los elementos de una lista: foldl (+) 0 lista
take/2	recibe un número "n" y una lista, devuelve los primeros "n" elementos de la lista. P.ej.: take 5 "abracadabra"
drop/2	recibe un número "n" y una lista, devuelve la lista a partir del elemento "n+1". P.ej.: drop 5 "abracadabra"

(1) el Hugs necesita mimitos para incorporar algunas funciones, p.ej. isAlpha. Para eso, cargar un programa que empiece así

```
module NombreDelArchivoHs where
import Hugs.Prelude
… acá mi programa …
```

p.ej. si el archivo se llama pruebas.hs, la primer línea es `module Pruebas where`.