```
 1:
 2:
 3:     [1]Chris Rathman / [2]ChrisRath@aol.com
 4:      ____________________________________________________________
 5:
 6: "************************************************************************
 7:  * Allowable characters:                                                 *
 8:  *    - a-z                                                             *
 9:  *    - A-Z                                                             *
10:  *    - 0-9                                                             *
11:  *    - .+/\*~<>@%|&?                                                   *
12:  *    - blank, tab, cr, ff, lf                                          *
13:  *                                                                      *
14:  * Variables:                                                           *
15:  *    - variables must be declared before use                           *
16:  *    - shared vars must begin with uppercase                           *
17:  *    - local vars must begin with lowercase                            *
18:  *    - reserved names: nil, true, false, self, super, and Smalltalk    *
19:  *                                                                      *
20:  * Variable scope:                                                      *
21:  *    - Global: defined in Dictionary Smalltalk and accessible by all   *
22:  *        objects in system                                             *
23:  *    - Special: (reserved) Smalltalk, super, self, true, false, & nil  *
24:  *    - Method Temporary: local to a method                             *
25:  *    - Block Temporary: local to a block                               *
26:  *    - Pool: variables in a Dictionary object                          *
27:  *    - Method Parameters: automatic local vars created as a result of  *
28:  *        message call with params                                      *
29:  *    - Block Parameters: automatic local vars created as a result of   *
30:  *        value: message call                                           *
31:  *    - Class: shared with all instances of one class & its subclasses  *
32:  *    - Class Instance: unique to each instance of a class              *
33:  *    - Instance Variables: unique to each instance                     *
34:  ************************************************************************"
35: "Comments are enclosed in quotes"
36: "Period (.) is the statement seperator"
37:
38: "************************************************************************
39:  * Transcript:                                                          *
40:  ************************************************************************"
41: Transcript clear.                                    "clear to transcript window"
42: Transcript show: 'Hello World'.                      "output string in transcript window"
43: Transcript nextPutAll: 'Hello World'.                "output string in transcript window"
44: Transcript nextPut: $A.                              "output character in transcript window"
45: Transcript space.                                    "output space character in transcript window"
```

```
46: Transcript tab.                                          "output tab character in transcript window"
47: Transcript cr.                                           "carriage return /linefeed"
48: 'Hello' printOn: Transcript.                             "append print string into the window"
49: 'Hello' storeOn: Transcript.                             "append store string into the window"
50: Transcript endEntry.                                     "flush the output buffer"
51:
52: "************************************************************************
53:  * Assignment:                                                           *
54:  ************************************************************************"
55: | x y |
56: x _ 4.                                                  "assignment (Squeak) <-"
57: x := 5.                                                 "assignment"
58: x := y := z := 6.                                       "compound assignment"
59: x := (y := 6) + 1.
60: x := Object new.                                        "bind to allocatedinstance of a class"
61: x := 123 class.                                         "discover the object class"
62: x := Integer superclass.                                "discover the superclass of a class"
63: x := Object allInstances.                               "get an array of all instances of a class"
64: x := Integer allSuperclasses.                           "get all superclasses of a class"
65: x := 1.2 hash.                                          "hash value for object"
66: y := x copy.                                            "copy object"
67: y := x shallowCopy.                                     "copy object (not overridden)"
68: y := x deepCopy.                                        "copy object and instance vars"
69: y := x veryDeepCopy.                                    "complete tree copy using a dictionary"
70:
71: "************************************************************************
72:  * Constants:                                                            *
73:  ************************************************************************"
74: | b |
75: b := true.                                             "true constant"
76: b := false.                                            "false constant"
77: x := nil.                                              "nil object constant"
78: x := 1.                                                "integer constants"
79: x := 3.14.                                             "float constants"
80: x := 2e-2.                                             "fractional constants"
81: x := 16r0F.                                            "hex constant".
82: x := -1.                                               "negative constants"
83: x := 'Hello'.                                          "string constant"
84: x := 'I''m here'.                                      "single quote escape"
85: x := $A.                                               "character constant"
86: x := $ .                                               "character constant (space)"
87: x := #aSymbol.                                         "symbol constants"
88: x := #(3 2 1).                                         "array constants"
89: x := #('abc' 2 $a).                                    "mixing of types allowed"
90:
```

```
 91: "**************************************************************************
 92:  * Booleans:                                                               *
 93:  **************************************************************************"
 94: | b x y |
 95: x := 1. y := 2.
 96: b := (x = y).                                       "equals"
 97: b := (x ~= y).                                      "not equals"
 98: b := (x == y).                                      "identical"
 99: b := (x ~~ y).                                      "not identical"
100: b := (x > y).                                       "greater than"
101: b := (x < y).                                       "less than"
102: b := (x >= y).                                      "greater than or equal"
103: b := (x <= y).                                      "less than or equal"
104: b := b not.                                         "boolean not"
105: b := (x < 5) & (y > 1).                             "boolean and"
106: b := (x < 5) | (y > 1).                             "boolean or"
107: b := (x < 5) and: [y > 1].                          "boolean and (short-circuit)"
108: b := (x < 5) or: [y > 1].                           "boolean or (short-circuit)"
109: b := (x < 5) eqv: (y > 1).                          "test if both trueor both false"
110: b := (x < 5) xor: (y > 1).                          "test if one true and other false"
111: b := 5 between: 3 and: 12.                          "between (inclusive)"
112: b := 123 isKindOf: Number.                          "test if object isclass or subclass of"
113: b := 123 isMemberOf: SmallInteger.                  "test if object istype of class"
114: b := 123 respondsTo: sqrt.                          "test if object responds to message"
115: b := x isNil.                                       "test if object isnil"
116: b := x isZero.                                      "test if number iszero"
117: b := x positive.                                    "test if number ispositive"
118: b := x strictlyPositive.                            "test if number isgreater than zero"
119: b := x negative.                                    "test if number isnegative"
120: b := x even.                                        "test if number iseven"
121: b := x odd.                                         "test if number isodd"
122: b := x isLiteral.                                   "test if literal constant"
123: b := x isInteger.                                   "test if object isinteger"
124: b := x isFloat.                                     "test if object isfloat"
125: b := x isNumber.                                    "test if object isnumber"
126: b := $A isUppercase.                                "test if upper case character"
127: b := $A isLowercase.                                "test if lower case character"
128:
129: "**************************************************************************
130:  * Arithmetic expressions:                                                 *
131:  **************************************************************************"
132: | x |
133: x := 6 + 3.                                        "addition"
134: x := 6 - 3.                                        "subtraction"
135: x := 6 * 3.                                        "multiplication"
```

```
136: x := 1 + 2 * 3.                    "evaluation alwaysleft to right (1 + 2) * 3"
137: x := 5 / 3.                        "division with fractional result"
138: x := 5.0 / 3.0.                    "division with float result"
139: x := 5.0 // 3.0.                   "integer divide"
140: x := 5.0 \\ 3.0.                   "integer remainder"
141: x := -5.                           "unary minus"
142: x := 5 sign.                       "numeric sign (1, -1 or 0)"
143: x := 5 negated.                    "negate receiver"
144: x := 1.2 integerPart.              "integer part of number (1.0)"
145: x := 1.2 fractionPart.             "fractional part of number (0.2)"
146: x := 5 reciprocal.                 "reciprocal function"
147: x := 6 * 3.1.                      "auto convert to float"
148: x := 5 squared.                    "square function"
149: x := 25 sqrt.                      "square root"
150: x := 5 raisedTo: 2.                "power function"
151: x := 5 raisedToInteger: 2.         "power function with integer"
152: x := 5 exp.                        "exponential"
153: x := -5 abs.                       "absolute value"
154: x := 3.99 rounded.                 "round"
155: x := 3.99 truncated.               "truncate"
156: x := 3.99 roundTo: 1.              "round to specified decimal places"
157: x := 3.99 truncateTo: 1.           "truncate to specified decimal places"
158: x := 3.99 floor.                   "truncate"
159: x := 3.99 ceiling.                 "round up"
160: x := 5 factorial.                  "factorial"
161: x := -5 quo: 3.                    "integer divide rounded toward zero"
162: x := -5 rem: 3.                    "integer remainderrounded toward zero"
163: x := 28 gcd: 12.                   "greatest common denominator"
164: x := 28 lcm: 12.                   "least common multiple"
165: x := 100 ln.                       "natural logarithm"
166: x := 100 log.                      "base 10 logarithm"
167: x := 100 log: 10.                  "logarithm with specified base"
168: x := 100 floorLog: 10.             "floor of the log"
169: x := 180 degreesToRadians.         "convert degrees to radians"
170: x := 3.14 radiansToDegrees.        "convert radians to degrees"
171: x := 0.7 sin.                      "sine"
172: x := 0.7 cos.                      "cosine"
173: x := 0.7 tan.                      "tangent"
174: x := 0.7 arcSin.                   "arcsine"
175: x := 0.7 arcCos.                   "arccosine"
176: x := 0.7 arcTan.                   "arctangent"
177: x := 10 max: 20.                   "get maximum of two numbers"
178: x := 10 min: 20.                   "get minimum of two numbers"
179: x := Float pi.                     "pi"
180: x := Float e.                      "exp constant"
```

```
181: x := Float infinity.                                          "infinity"
182: x := Float nan.                                               "not-a-number"
183: x := Random new next; yourself. x next.                       "random number stream (0.0 to 1.0)
184: x := 100 atRandom.                                            "quick random number"
185:
186: "*************************************************************************
187:  * Bitwise Manipulation:                                                  *
188:  *************************************************************************"
189: | b x |
190: x := 16rFF bitAnd: 16r0F.                                     "and bits"
191: x := 16rF0 bitOr: 16r0F.                                      "or bits"
192: x := 16rFF bitXor: 16r0F.                                     "xor bits"
193: x := 16rFF bitInvert.                                         "invert bits"
194: x := 16r0F bitShift: 4.                                       "left shift"
195: x := 16rF0 bitShift: -4.                                      "right shift"
196: "x := 16r80 bitAt: 7."                                        "bit at position (0|1) [!Squeak]"
197: x := 16r80 highbit.                                           "position of highest bit set"
198: b := 16rFF allMask: 16r0F.                                    "test if all bits set in mask set in receiver"
199: b := 16rFF anyMask: 16r0F.                                    "test if any bits set in mask set in receiver"
200: b := 16rFF noMask: 16r0F.                                     "test if all bits set in mask clear in receiver"
201:
202: "*************************************************************************
203:  * Conversion:                                                            *
204:  *************************************************************************"
205: | x |
206: x := 3.99 asInteger.                                         "convert number tointeger (truncates in Squeak)"
207: x := 3.99 asFraction.                                        "convert number tofraction"
208: x := 3 asFloat.                                              "convert number tofloat"
209: x := 65 asCharacter.                                         "convert integer to character"
210: x := $A asciiValue.                                          "convert characterto integer"
211: x := 3.99 printString.                                       "convert object tostring via printOn:"
212: x := 3.99 storeString.                                       "convert object tostring via storeOn:"
213: x := 15 radix: 16.                                           "convert to stringin given base"
214: x := 15 printStringBase: 16.
215: x := 15 storeStringBase: 16.
216:
217: "*************************************************************************
218:  * Blocks:                                                                *
219:  *    - blocks are objects and may be assigned to a variable              *
220:  *    - value is last expression evaluated unless explicit return         *
221:  *    - blocks may be nested                                              *
222:  *    - specification [ arguments | | localvars | expressions ]           *
223:  *    - Squeak does not currently support localvars in blocks             *
224:  *    - max of three arguments allowed                                    *
225:  *    - ^expression terminates block & method (exits all nested blocks)   *
```

```
226:  *    - blocks intended for long term storage should not contain ^      *
227:  ************************************************************************"
228: | x y z |
229: x := [ y := 1. z := 2. ]. x value.                          "simple block usage"
230: x := [ :argOne :argTwo |   argOne, ' and ' , argTwo.].      "set up block withargument passing"
231: Transcript show: (x value: 'First' value: 'Second'); cr.    "use block with argument passing"
232: "x := [ | z | z := 1.].                                     localvars not available in squeak blocks"
233:
234: "************************************************************************
235:  * Method calls:                                                        *
236:  *    - unary methods are messages with no arguments                    *
237:  *    - binary methods                                                  *
238:  *    - keyword methods are messages with selectors including colons    *
239:  *                                                                      *
240:  * standard categories/protocols:                                       *
241:  *    - initialize-release    (methods called for new instance)         *
242:  *    - accessing             (get/set methods)                         *
243:  *    - testing               (boolean tests - is)                      *
244:  *    - comparing             (boolean tests with parameter             *
245:  *    - displaying            (gui related methods)                     *
246:  *    - printing              (methods for printing)                    *
247:  *    - updating              (receive notification of changes)         *
248:  *    - private               (methods private to class)                *
249:  *    - instance-creation     (class methods for creating instance)     *
250:  ************************************************************************"
251: | x |
252: x := 2 sqrt.                                                 "unary message"
253: x := 2 raisedTo: 10.                                         "keyword message"
254: x := 194 * 9.                                               "binary message"
255: Transcript show: (194 * 9) printString; cr.                  "combination (chaining)"
256: x := 2 perform: #sqrt.                                      "indirect method invocation"
257: Transcript                                                  "Cascading - send multiple messages to receiver"
258:    show: 'hello ';
259:    show: 'world';
260:    cr.
261: x := 3 + 2; * 100.                                          "result=300. Sendsmessage to same receiver (3)"
262:
263: "************************************************************************
264:  * Conditional Statements:                                             *
265:  ************************************************************************"
266: | x |
267: x > 10 ifTrue: [Transcript show: 'ifTrue'; cr].             "if then"
268: x > 10 ifFalse: [Transcript show: 'ifFalse'; cr].           "if else"
269: x > 10                                                      "if then else"
270:    ifTrue: [Transcript show: 'ifTrue'; cr]
```

```
271:    ifFalse: [Transcript show: 'ifFalse'; cr].
272: x > 10                                                    "if else then"
273:    ifFalse: [Transcript show: 'ifFalse'; cr]
274:    ifTrue: [Transcript show: 'ifTrue'; cr].
275: Transcript
276:    show:
277:       (x > 10
278:          ifTrue: ['ifTrue']
279:          ifFalse: ['ifFalse']);
280:    cr.
281: Transcript                                                 "nested if then else"
282:    show:
283:       (x > 10
284:          ifTrue: [x > 5
285:             ifTrue: ['A']
286:             ifFalse: ['B']]
287:          ifFalse: ['C']);
288:    cr.
289: switch := Dictionary new.                                  "switch functionality"
290: switch at: $A put: [Transcript show: 'Case A'; cr].
291: switch at: $B put: [Transcript show: 'Case B'; cr].
292: switch at: $C put: [Transcript show: 'Case C'; cr].
293: result := (switch at: $B) value.
294:
295: "************************************************************************
296:  * Iteration statements:                                                *
297:  ************************************************************************"
298: | x y |
299: x := 4. y := 1.
300: [x > 0] whileTrue: [x := x - 1. y := y * 2].               "while true loop"
301: [x >= 4] whileFalse: [x := x + 1. y := y * 2].            "while false loop"
302: x timesRepeat: [y := y * 2].                              "times repear loop(i := 1 to x)"
303: 1 to: x do: [:a | y := y * 2].                            "for loop"
304: 1 to: x by: 2 do: [:a | y := y / 2].                      "for loop with specified increment"
305: #(5 4 3) do: [:a | x := x + a].                           "iterate over array elements"
306:
307: "************************************************************************
308:  * Character:                                                           *
309:  ************************************************************************"
310: | x y |
311: x := $A.                                                  "character assignment"
312: y := x isLowercase.                                       "test if lower case"
313: y := x isUppercase.                                       "test if upper case"
314: y := x isLetter.                                          "test if letter"
315: y := x isDigit.                                           "test if digit"
```

```
316: y := x isAlphaNumeric.                                    "test if alphanumeric"
317: y := x isSeparator.                                       "test if seperatorchar"
318: y := x isVowel.                                           "test if vowel"
319: y := x digitValue.                                        "convert to numeric digit value"
320: y := x asLowercase.                                       "convert to lower case"
321: y := x asUppercase.                                       "convert to upper case"
322: y := x asciiValue.                                        "convert to numeric ascii value"
323: y := x asString.                                          "convert to string"
324: b := $A <= $B.                                            "comparison"
325: y := $A max: $B.
326:
327: "************************************************************************
328:  * Symbol:                                                               *
329:  ************************************************************************"
330: | b x y |
331: x := #Hello.                                              "symbol assignment"
332: y := 'String', 'Concatenation'.                           "symbol concatenation (result is string)"
333: b := x isEmpty.                                           "test if symbol isempty"
334: y := x size.                                              "string size"
335: y := x at: 2.                                             "char at location"
336: y := x copyFrom: 2 to: 4.                                 "substring"
337: y := x indexOf: $e ifAbsent: [0].                         "first position ofcharacter within string"
338: x do: [:a | Transcript show: a printString; cr].          "iterate over the string"
339: b := x conform: [:a | (a >= $a) & (a <= $z)].             "test if all elements meet condition"
340: y := x select: [:a | a > $a].                             "return all elements that meet condition"
341: y := x asString.                                          "convert symbol tostring"
342: y := x asText.                                            "convert symbol totext"
343: y := x asArray.                                           "convert symbol toarray"
344: y := x asOrderedCollection.                               "convert symbol toordered collection"
345: y := x asSortedCollection.                                "convert symbol tosorted collection"
346: y := x asBag.                                             "convert symbol tobag collection"
347: y := x asSet.                                             "convert symbol toset collection"
348:
349: "************************************************************************
350:  * String:                                                               *
351:  ************************************************************************"
352: | b x y |
353: x := 'This is a string'.                                 "string assignment"
354: x := 'String', 'Concatenation'.                           "string concatenation"
355: b := x isEmpty.                                           "test if string isempty"
356: y := x size.                                              "string size"
357: y := x at: 2.                                             "char at location"
358: y := x copyFrom: 2 to: 4.                                 "substring"
359: y := x indexOf: $a ifAbsent: [0].                         "first position ofcharacter within string"
360: x := String new: 4.                                       "allocate string object"
```

```
x                                                       "set string elements"
   at: 1 put: $a;
   at: 2 put: $b;
   at: 3 put: $c;
   at: 4 put: $e.
x := String with: $a with: $b with: $c with: $d.        "set up to 4 elements at a time"
x do: [:a | Transcript show: a printString; cr].        "iterate over the string"
b := x conform: [:a | (a >= $a) & (a <= $z)].           "test if all elements meet condition"
y := x select: [:a | a > $a].                           "return all elements that meet condition"
y := x asSymbol.                                        "convert string tosymbol"
y := x asArray.                                         "convert string toarray"
x := 'ABCD' asByteArray.                                "convert string tobyte array"
y := x asOrderedCollection.                             "convert string toordered collection"
y := x asSortedCollection.                              "convert string tosorted collection"
y := x asBag.                                           "convert string tobag collection"
y := x asSet.                                           "convert string toset collection"
y := x shuffled.                                        "randomly shuffle string"

"************************************************************************
 * Array:         Fixed length collection                                *
 * ByteArray:     Array limited to byte elements (0-255)                 *
 * WordArray:     Array limited to word elements (0-2^32)                *
 ************************************************************************"
| b x y sum max |
x := #(4 3 2 1).                                        "constant array"
x := Array with: 5 with: 4 with: 3 with: 2.             "create array withup to 4 elements"
x := Array new: 4.                                      "allocate an arraywith specified size"
x                                                       "set array elements"
   at: 1 put: 5;
   at: 2 put: 4;
   at: 3 put: 3;
   at: 4 put: 2.
b := x isEmpty.                                         "test if array is empty"
y := x size.                                            "array size"
y := x at: 4.                                           "get array elementat index"
b := x includes: 3.                                     "test if element is in array"
y := x copyFrom: 2 to: 4.                               "subarray"
y := x indexOf: 3 ifAbsent: [0].                        "first position ofelement within array"
y := x occurrencesOf: 3.                                "number of times object in collection"
x do: [:a | Transcript show: a printString; cr].        "iterate over the array"
b := x conform: [:a | (a >= 1) & (a <= 4)].             "test if all elements meet condition"
y := x select: [:a | a > 2].                            "return collectionof elements that pass test"
y := x reject: [:a | a < 2].                            "return collectionof elements that fail test"
y := x collect: [:a | a + a].                           "transform each element for new collection"
y := x detect: [:a | a > 3] ifNone: [].                 "find position of first element that passes test"
```

```
406: sum := 0. x do: [:a | sum := sum + a]. sum.                "sum array elements"
407: sum := 0. 1 to: (x size) do: [:a | sum := sum + (x at: a)]. "sum array elements"
408: sum := x inject: 0 into: [:a :c | a + c].                  "sum array elements"
409: max := x inject: 0 into: [:a :c | (a > c)                  "find max element in array"
410:    ifTrue: [a]
411:    ifFalse: [c]].
412: y := x shuffled.                                           "randomly shuffle collection"
413: y := x asArray.                                            "convert to array"
414: "y := x asByteArray."                                      "note: this instruction not available on Squeak"
415: y := x asWordArray.                                        "convert to word array"
416: y := x asOrderedCollection.                                "convert to ordered collection"
417: y := x asSortedCollection.                                 "convert to sortedcollection"
418: y := x asBag.                                              "convert to bag collection"
419: y := x asSet.                                              "convert to set collection"
420:
421: "************************************************************************
422:  * OrderedCollection: acts like an expandable array                      *
423:  ************************************************************************"
424: | b x y sum max |
425: x := OrderedCollection with: 4 with: 3 with: 2 with: 1.    "create collectionwith up to 4 elements"
426: x := OrderedCollection new.                                "allocate collection"
427: x add: 3; add: 2; add: 1; add: 4; yourself.                "add element to collection"
428: y := x addFirst: 5.                                        "add element at beginning of collection"
429: y := x removeFirst.                                        "remove first element in collection"
430: y := x addLast: 6.                                         "add element at end of collection"
431: y := x removeLast.                                         "remove last element in collection"
432: y := x addAll: #(7 8 9).                                   "add multiple elements to collection"
433: y := x removeAll: #(7 8 9).                                "remove multiple elements from collection"
434: x at: 2 put: 3.                                            "set element at index"
435: y := x remove: 5 ifAbsent: [].                             "remove element from collection"
436: b := x isEmpty.                                            "test if empty"
437: y := x size.                                               "number of elements"
438: y := x at: 2.                                              "retrieve element at index"
439: y := x first.                                              "retrieve first element in collection"
440: y := x last.                                               "retrieve last element in collection"
441: b := x includes: 5.                                        "test if element is in collection"
442: y := x copyFrom: 2 to: 3.                                  "subcollection"
443: y := x indexOf: 3 ifAbsent: [0].                           "first position ofelement within collection"
444: y := x occurrencesOf: 3.                                   "number of times object in collection"
445: x do: [:a | Transcript show: a printString; cr].           "iterate over the collection"
446: b := x conform: [:a | (a >= 1) & (a <= 4)].                "test if all elements meet condition"
447: y := x select: [:a | a > 2].                               "return collectionof elements that pass test"
448: y := x reject: [:a | a < 2].                               "return collectionof elements that fail test"
449: y := x collect: [:a | a + a].                              "transform each element for new collection"
450: y := x detect: [:a | a > 3] ifNone: [].                    "find position of first element that passes test"
```

```
451: sum := 0. x do: [:a | sum := sum + a]. sum.                  "sum elements"
452: sum := 0. 1 to: (x size) do: [:a | sum := sum + (x at: a)]. "sum elements"
453: sum := x inject: 0 into: [:a :c | a + c].                    "sum elements"
454: max := x inject: 0 into: [:a :c | (a > c)                    "find max element in collection"
455:    ifTrue: [a]
456:    ifFalse: [c]].
457: y := x shuffled.                                            "randomly shuffle collection"
458: y := x asArray.                                             "convert to array"
459: y := x asOrderedCollection.                                 "convert to ordered collection"
460: y := x asSortedCollection.                                  "convert to sortedcollection"
461: y := x asBag.                                               "convert to bag collection"
462: y := x asSet.                                               "convert to set collection"
463:
464: "************************************************************************
465:  * SortedCollection:    like OrderedCollection except order of elements *
466:  *                          determined by sorting criteria               *
467:  ************************************************************************"
468: | b x y sum max |
469: x := SortedCollection with: 4 with: 3 with: 2 with: 1.      "create collectionwith up to 4 elements"
470: x := SortedCollection new.                                  "allocate collection"
471: x := SortedCollection sortBlock: [:a :c | a > c].           "set sort criteria"
472: x add: 3; add: 2; add: 1; add: 4; yourself.                 "add element to collection"
473: y := x addFirst: 5.                                         "add element at beginning of collection"
474: y := x removeFirst.                                         "remove first element in collection"
475: y := x addLast: 6.                                          "add element at end of collection"
476: y := x removeLast.                                          "remove last element in collection"
477: y := x addAll: #(7 8 9).                                    "add multiple elements to collection"
478: y := x removeAll: #(7 8 9).                                 "remove multiple elements from collection"
479: y := x remove: 5 ifAbsent: [].                              "remove element from collection"
480: b := x isEmpty.                                             "test if empty"
481: y := x size.                                                "number of elements"
482: y := x at: 2.                                               "retrieve element at index"
483: y := x first.                                               "retrieve first element in collection"
484: y := x last.                                                "retrieve last element in collection"
485: b := x includes: 4.                                         "test if element is in collection"
486: y := x copyFrom: 2 to: 3.                                   "subcollection"
487: y := x indexOf: 3 ifAbsent: [0].                            "first position ofelement within collection"
488: y := x occurrencesOf: 3.                                    "number of times object in collection"
489: x do: [:a | Transcript show: a printString; cr].            "iterate over the collection"
490: b := x conform: [:a | (a >= 1) & (a <= 4)].                 "test if all elements meet condition"
491: y := x select: [:a | a > 2].                                "return collectionof elements that pass test"
492: y := x reject: [:a | a < 2].                                "return collectionof elements that fail test"
493: y := x collect: [:a | a + a].                               "transform each element for new collection"
494: y := x detect: [:a | a > 3] ifNone: [].                     "find position of first element that passes test"
495: sum := 0. x do: [:a | sum := sum + a]. sum.                 "sum elements"
```

```
496: sum := 0. 1 to: (x size) do: [:a | sum := sum + (x at: a)]. "sum elements"
497: sum := x inject: 0 into: [:a :c | a + c].                   "sum elements"
498: max := x inject: 0 into: [:a :c | (a > c)                   "find max element in collection"
499:    ifTrue: [a]
500:    ifFalse: [c]].
501: y := x asArray.                                            "convert to array"
502: y := x asOrderedCollection.                                "convert to ordered collection"
503: y := x asSortedCollection.                                 "convert to sortedcollection"
504: y := x asBag.                                              "convert to bag collection"
505: y := x asSet.                                              "convert to set collection"
506:
507: "*************************************************************************
508:  * Bag:        like OrderedCollection except elements are in no         *
509:  *                particular order                                       *
510:  *************************************************************************"
511: | b x y sum max |
512: x := Bag with: 4 with: 3 with: 2 with: 1.                  "create collectionwith up to 4 elements"
513: x := Bag new.                                              "allocate collection"
514: x add: 4; add: 3; add: 1; add: 2; yourself.                "add element to collection"
515: x add: 3 withOccurrences: 2.                               "add multiple copies to collection"
516: y := x addAll: #(7 8 9).                                   "add multiple elements to collection"
517: y := x removeAll: #(7 8 9).                                "remove multiple elements from collection"
518: y := x remove: 4 ifAbsent: [].                             "remove element from collection"
519: b := x isEmpty.                                            "test if empty"
520: y := x size.                                               "number of elements"
521: b := x includes: 3.                                        "test if element is in collection"
522: y := x occurrencesOf: 3.                                   "number of times object in collection"
523: x do: [:a | Transcript show: a printString; cr].           "iterate over the collection"
524: b := x conform: [:a | (a >= 1) & (a <= 4)].                "test if all elements meet condition"
525: y := x select: [:a | a > 2].                               "return collectionof elements that pass test"
526: y := x reject: [:a | a < 2].                               "return collectionof elements that fail test"
527: y := x collect: [:a | a + a].                              "transform each element for new collection"
528: y := x detect: [:a | a > 3] ifNone: [].                    "find position of first element that passes test"
529: sum := 0. x do: [:a | sum := sum + a]. sum.                "sum elements"
530: sum := x inject: 0 into: [:a :c | a + c].                  "sum elements"
531: max := x inject: 0 into: [:a :c | (a > c)                  "find max element in collection"
532:    ifTrue: [a]
533:    ifFalse: [c]].
534: y := x asOrderedCollection.                                "convert to ordered collection"
535: y := x asSortedCollection.                                 "convert to sortedcollection"
536: y := x asBag.                                              "convert to bag collection"
537: y := x asSet.                                              "convert to set collection"
538:
539: "*************************************************************************
540:  * Set:          like Bag except duplicates not allowed                  *
```

```
541:  * IdentitySet:   uses identity test (== rather than =)                 *
542:  ************************************************************************"
543: | b x y sum max |
544: x := Set with: 4 with: 3 with: 2 with: 1.                  "create collectionwith up to 4 elements"
545: x := Set new.                                              "allocate collection"
546: x add: 4; add: 3; add: 1; add: 2; yourself.                "add element to collection"
547: y := x addAll: #(7 8 9).                                   "add multiple elements to collection"
548: y := x removeAll: #(7 8 9).                                "remove multiple elements from collection"
549: y := x remove: 4 ifAbsent: [].                             "remove element from collection"
550: b := x isEmpty.                                            "test if empty"
551: y := x size.                                               "number of elements"
552: x includes: 4.                                             "test if element is in collection"
553: x do: [:a | Transcript show: a printString; cr].           "iterate over the collection"
554: b := x conform: [:a | (a >= 1) & (a <= 4)].                "test if all elements meet condition"
555: y := x select: [:a | a > 2].                               "return collectionof elements that pass test"
556: y := x reject: [:a | a < 2].                               "return collectionof elements that fail test"
557: y := x collect: [:a | a + a].                              "transform each element for new collection"
558: y := x detect: [:a | a > 3] ifNone: [].                    "find position of first element that passes test"
559: sum := 0. x do: [:a | sum := sum + a]. sum.                "sum elements"
560: sum := x inject: 0 into: [:a :c | a + c].                  "sum elements"
561: max := x inject: 0 into: [:a :c | (a > c)                  "find max element in collection"
562:    ifTrue: [a]
563:    ifFalse: [c]].
564: y := x asArray.                                            "convert to array"
565: y := x asOrderedCollection.                                "convert to ordered collection"
566: y := x asSortedCollection.                                 "convert to sortedcollection"
567: y := x asBag.                                              "convert to bag collection"
568: y := x asSet.                                              "convert to set collection"
569:
570: "************************************************************************
571:  * Interval:                                                            *
572:  ************************************************************************"
573: | b x y sum max |
574: x := Interval from: 5 to: 10.                               "create interval object"
575: x := 5 to: 10.
576: x := Interval from: 5 to: 10 by: 2.                        "create interval object with specified increment"
577: x := 5 to: 10 by: 2.
578: b := x isEmpty.                                            "test if empty"
579: y := x size.                                               "number of elements"
580: x includes: 9.                                             "test if element is in collection"
581: x do: [:k | Transcript show: k printString; cr].           "iterate over interval"
582: b := x conform: [:a | (a >= 1) & (a <= 4)].                "test if all elements meet condition"
583: y := x select: [:a | a > 7].                               "return collectionof elements that pass test"
584: y := x reject: [:a | a < 2].                               "return collectionof elements that fail test"
585: y := x collect: [:a | a + a].                              "transform each element for new collection"
```

```
y := x detect: [:a | a > 3] ifNone: [].                    "find position of first element that passes test"
sum := 0. x do: [:a | sum := sum + a]. sum.                "sum elements"
sum := 0. 1 to: (x size) do: [:a | sum := sum + (x at: a)]. "sum elements"
sum := x inject: 0 into: [:a :c | a + c].                  "sum elements"
max := x inject: 0 into: [:a :c | (a > c)                  "find max element in collection"
   ifTrue: [a]
   ifFalse: [c]].
y := x asArray.                                            "convert to array"
y := x asOrderedCollection.                                "convert to ordered collection"
y := x asSortedCollection.                                 "convert to sortedcollection"
y := x asBag.                                              "convert to bag collection"
y := x asSet.                                              "convert to set collection"

"************************************************************************
 * Associations:                                                        *
 ************************************************************************"
| x y |
x := #myVar->'hello'.
y := x key.
y := x value.

"************************************************************************
 * Dictionary:                                                          *
 * IdentityDictionary:   uses identity test (== rather than =)          *
 ************************************************************************"
| b x y |
x := Dictionary new.                                       "allocate collection"
x add: #a->4; add: #b->3; add: #c->1; add: #d->2; yourself. "add element to collection"
x at: #e put: 3.                                           "set element at index"
b := x isEmpty.                                            "test if empty"
y := x size.                                               "number of elements"
y := x at: #a ifAbsent: [].                                "retrieve element at index"
y := x keyAtValue: 3 ifAbsent: [].                         "retrieve key for given value with error block"
y := x removeKey: #e ifAbsent: [].                         "remove element from collection"
b := x includes: 3.                                        "test if element is in values collection"
b := x includesKey: #a.                                    "test if element is in keys collection"
y := x occurrencesOf: 3.                                   "number of times object in collection"
y := x keys.                                               "set of keys"
y := x values.                                             "bag of values"
x do: [:a | Transcript show: a printString; cr].           "iterate over the values collection"
x keysDo: [:a | Transcript show: a printString; cr].       "iterate over the keys collection"
x associationsDo: [:a | Transcript show: a printString; cr]."iterate over the associations"
x keysAndValuesDo: [:aKey :aValue | Transcript             "iterate over keysand values"
   show: aKey printString; space;
   show: aValue printString; cr].
```

```
631: b := x conform: [:a | (a >= 1) & (a <= 4)].                "test if all elements meet condition"
632: y := x select: [:a | a > 2].                               "return collectionof elements that pass test"
633: y := x reject: [:a | a < 2].                               "return collectionof elements that fail test"
634: y := x collect: [:a | a + a].                              "transform each element for new collection"
635: y := x detect: [:a | a > 3] ifNone: [].                    "find position of first element that passes test"
636: sum := 0. x do: [:a | sum := sum + a]. sum.                "sum elements"
637: sum := x inject: 0 into: [:a :c | a + c].                  "sum elements"
638: max := x inject: 0 into: [:a :c | (a > c)                  "find max element in collection"
639:    ifTrue: [a]
640:    ifFalse: [c]].
641: y := x asArray.                                            "convert to array"
642: y := x asOrderedCollection.                                "convert to ordered collection"
643: y := x asSortedCollection.                                 "convert to sortedcollection"
644: y := x asBag.                                              "convert to bag collection"
645: y := x asSet.                                              "convert to set collection"
646:
647: Smalltalk at: #CMRGlobal put: 'CMR entry'.                  "put global in Smalltalk Dictionary"
648: x := Smalltalk at: #CMRGlobal.                             "read global from Smalltalk Dictionary"
649: Transcript show: (CMRGlobal printString).                  "entries are directly accessible by name"
650: Smalltalk keys do: [ :k |                                  "print out all classes"
651:    ((Smalltalk at: k) isKindOf: Class)
652:       ifFalse: [Transcript show: k printString; cr]].
653: Smalltalk at: #CMRDictionary put: (Dictionary new).         "set up user defined dictionary"
654: CMRDictionary at: #MyVar1 put: 'hello1'.                    "put entry in dictionary"
655: CMRDictionary add: #MyVar2->'hello2'.                      "add entry to dictionary use key->value combo"
656: CMRDictionary size.                                        "dictionary size"
657: CMRDictionary keys do: [ :k |                              "print out keys indictionary"
658:    Transcript show: k printString; cr].
659: CMRDictionary values do: [ :k |                            "print out values in dictionary"
660:    Transcript show: k printString; cr].
661: CMRDictionary keysAndValuesDo: [:aKey :aValue |            "print out keys and values"
662:    Transcript
663:       show: aKey printString;
664:       space;
665:       show: aValue printString;
666:       cr].
667: CMRDictionary associationsDo: [:aKeyValue |                "another iterator for printing key values"
668:    Transcript show: aKeyValue printString; cr].
669: Smalltalk removeKey: #CMRGlobal ifAbsent: [].              "remove entry fromSmalltalk dictionary"
670: Smalltalk removeKey: #CMRDictionary ifAbsent: [].          "remove user dictionary from Smalltalk dictionary"
671:
672: "*********************************************************************
673:  * Internal Stream:                                                  *
674:  *********************************************************************"
675: | b x ios |
```

```
676: ios := ReadStream on: 'Hello read stream'.
677: ios := ReadStream on: 'Hello read stream' from: 1 to: 5.
678: [(x := ios nextLine) notNil]
679:    whileTrue: [Transcript show: x; cr].
680: ios position: 3.
681: ios position.
682: x := ios next.
683: x := ios peek.
684: x := ios contents.
685: b := ios atEnd.
686:
687: ios := ReadWriteStream on: 'Hello read stream'.
688: ios := ReadWriteStream on: 'Hello read stream' from: 1 to: 5.
689: ios := ReadWriteStream with: 'Hello read stream'.
690: ios := ReadWriteStream with: 'Hello read stream' from: 1 to: 10.
691: ios position: 0.
692: [(x := ios nextLine) notNil]
693:    whileTrue: [Transcript show: x; cr].
694: ios position: 6.
695: ios position.
696: ios nextPutAll: 'Chris'.
697: x := ios next.
698: x := ios peek.
699: x := ios contents.
700: b := ios atEnd.
701:
702: "*************************************************************************
703:  * FileStream:                                                           *
704:  *************************************************************************"
705: | b x ios |
706: ios := FileStream newFileNamed: 'ios.txt'.
707: ios nextPut: $H; cr.
708: ios nextPutAll: 'Hello File'; cr.
709: 'Hello File' printOn: ios.
710: 'Hello File' storeOn: ios.
711: ios close.
712:
713: ios := FileStream oldFileNamed: 'ios.txt'.
714: [(x := ios nextLine) notNil]
715:    whileTrue: [Transcript show: x; cr].
716: ios position: 3.
717: x := ios position.
718: x := ios next.
719: x := ios peek.
720: b := ios atEnd.
```

```
721: ios close.
722:
723: "************************************************************************
724:  * Date:                                                                 *
725:  ************************************************************************"
726: | x y |
727: x := Date today.                                          "create date for today"
728: x := Date dateAndTimeNow.                                 "create date from current time/date"
729: x := Date readFromString: '01/02/1999'.                   "create date from formatted string"
730: x := Date newDay: 12 month: #July year: 1999              "create date from parts"
731: x := Date fromDays: 36000.                                "create date from elapsed days since 1/1/1901"
732: y := Date dayOfWeek: #Monday.                             "day of week as int (1-7)"
733: y := Date indexOfMonth: #January.                         "month of year as int (1-12)"
734: y := Date daysInMonth: 2 forYear: 1996.                   "day of month as int (1-31)"
735: y := Date daysInYear: 1996.                               "days in year (365|366)"
736: y := Date nameOfDay: 1                                    "weekday name (#Monday,...)"
737: y := Date nameOfMonth: 1.                                 "month name (#January,...)"
738: y := Date leapYear: 1996.                                 "1 if leap year; 0if not leap year"
739: y := x weekday.                                           "day of week (#Monday,...)"
740: y := x previous: #Monday.                                 "date for previousday of week"
741: y := x dayOfMonth.                                        "day of month (1-31)"
742: y := x day.                                               "day of year (1-366)"
743: y := x firstDayOfMonth.                                   "day of year for first day of month"
744: y := x monthName.                                         "month of year (#January,...)"
745: y := x monthIndex.                                        "month of year (1-12)"
746: y := x daysInMonth.                                       "days in month (1-31)"
747: y := x year.                                              "year (19xx)"
748: y := x daysInYear.                                        "days in year (365|366)"
749: y := x daysLeftInYear.                                    "days left in year(364|365)"
750: y := x asSeconds.                                         "seconds elapsed since 1/1/1901"
751: y := x addDays: 10.                                       "add days to date object"
752: y := x subtractDays: 10.                                  "subtract days to date object"
753: y := x subtractDate: (Date today).                        "subtract date (result in days)"
754: y := x printFormat: #(2 1 3 $/ 1 1).                      "print formatted date"
755: b := (x <= Date today).                                   "comparison"
756:
757: "************************************************************************
758:  * Time:                                                                 *
759:  ************************************************************************"
760: | x y |
761: x := Time now.                                           "create time from current time"
762: x := Time dateAndTimeNow.                                 "create time from current time/date"
763: x := Time readFromString: '3:47:26 pm'.                   "create time from formatted string"
764: x := Time fromSeconds: (60 * 60 * 4).                     "create time from elapsed time from midnight"
765: y := Time millisecondClockValue.                          "milliseconds since midnight"
```

```
766: y := Time totalSeconds.                                    "total seconds since 1/1/1901"
767: y := x seconds.                                            "seconds past minute (0-59)"
768: y := x minutes.                                            "minutes past hour(0-59)"
769: y := x hours.                                              "hours past midnight (0-23)"
770: y := x addTime: (Time now).                                "add time to time object"
771: y := x subtractTime: (Time now).                           "subtract time to time object"
772: y := x asSeconds.                                          "convert time to seconds"
773: x := Time millisecondsToRun: [                             "timing facility"
774:    1 to: 1000 do: [:index | y := 3.14 * index]].
775: b := (x <= Time now).                                      "comparison"
776:
777: "************************************************************************
778:  * Point:                                                               *
779:  ************************************************************************"
780: | x y |
781: x := 200@100.                                              "obtain a new point"
782: y := x x.                                                  "x coordinate"
783: y := x y.                                                  "y coordinate"
784: x := 200@100 negated.                                     "negates x and y"
785: x := (-200@-100) abs.                                     "absolute value ofx and y"
786: x := (200.5@100.5) rounded.                               "round x and y"
787: x := (200.5@100.5) truncated.                             "truncate x and y"
788: x := 200@100 + 100.                                       "add scale to bothx and y"
789: x := 200@100 - 100.                                       "subtract scale from both x and y"
790: x := 200@100 * 2.                                         "multiply x and y by scale"
791: x := 200@100 / 2.                                         "divide x and y byscale"
792: x := 200@100 // 2.                                        "divide x and y byscale"
793: x := 200@100 \\ 3.                                        "remainder of x and y by scale"
794: x := 200@100 + 50@25.                                     "add points"
795: x := 200@100 - 50@25.                                     "subtract points"
796: x := 200@100 * 3@4.                                       "multiply points"
797: x := 200@100 // 3@4.                                      "divide points"
798: x := 200@100 max: 50@200.                                 "max x and y"
799: x := 200@100 min: 50@200.                                 "min x and y"
800: x := 20@5 dotProduct: 10@2.                               "sum of product (x1*x2 + y1*y2)"
801:
802: "************************************************************************
803:  * Rectangle:                                                           *
804:  ************************************************************************"
805: Rectangle fromUser.
806:
807: "************************************************************************
808:  * Pen:                                                                 *
809:  ************************************************************************"
810: | myPen |
```

```
811: Display restoreAfter: [
812:    Display fillWhite.
813:
814: myPen := Pen new.                                          "get graphic pen"
815: myPen squareNib: 1.
816: myPen color: (Color blue).                                 "set pen color"
817: myPen home.                                                "position pen at center of display"
818: myPen up.                                                  "makes nib unable to draw"
819: myPen down.                                                "enable the nib todraw"
820: myPen north.                                               "points direction towards top"
821: myPen turn: -180.                                          "add specified degrees to direction"
822: myPen direction.                                           "get current angleof pen"
823: myPen go: 50.                                              "move pen specified number of pixels"
824: myPen location.                                            "get the pen position"
825: myPen goto: 200@200.                                       "move to specifiedpoint"
826: myPen place: 250@250.                                      "move to specifiedpoint without drawing"
827: myPen print: 'Hello World' withFont: (TextStyle default fontAt: 1).
828: Display extent.                                            "get display width@height"
829: Display width.                                             "get display width"
830: Display height.                                            "get display height"
831:
832: ].
833:
834: "************************************************************************
835:  * Dynamic Message Calling/Compiling:                                    *
836:  ************************************************************************"
837: | receiver message result argument keyword1 keyword2 argument1 argument2 |
838: "unary message"
839: receiver := 5.
840: message := 'factorial' asSymbol.
841: result := receiver perform: message.
842: result := Compiler evaluate: ((receiver storeString), ' ', message).
843: result := (Message new setSelector: message arguments: #()) sentTo: receiver.
844:
845: "binary message"
846: receiver := 1.
847: message := '+' asSymbol.
848: argument := 2.
849: result := receiver perform: message withArguments: (Array with: argument).
850: result := Compiler evaluate: ((receiver storeString), ' ', message, ' ', (argument storeString)).
851: result := (Message new setSelector: message arguments: (Array with: argument))sentTo: receiver.
852:
853: "keyword messages"
854: receiver := 12.
855: keyword1 := 'between:' asSymbol.
```

```
856: keyword2 := 'and:' asSymbol.
857: argument1 := 10.
858: argument2 := 20.
859: result := receiver
860:    perform: (keyword1, keyword2) asSymbol
861:    withArguments: (Array with: argument1 with: argument2).
862: result := Compiler evaluate:
863:    ((receiver storeString), ' ', keyword1, (argument1 storeString) , ' ', keyword2, (argument2 storeString)).
864: result := (Message
865:    new
866:       setSelector: (keyword1, keyword2) asSymbol
867:       arguments: (Array with: argument1 with: argument2))
868:    sentTo: receiver.
869:
870: "************************************************************************
871:  * class/meta-class:                                                    *
872:  ************************************************************************"
873: | b x |
874: x := String name.                                          "class name"
875: x := String category.                                      "organization category"
876: x := String comment.                                       "class comment"
877: x := String kindOfSubclass.                               "subclass type - subclass: variableSubclass, etc"
878: x := String definition.                                   "class definition"
879: x := String instVarNames.                                 "immediate instance variable names"
880: x := String allInstVarNames.                              "accumulated instance variable names"
881: x := String classVarNames.                                "immediate class variable names"
882: x := String allClassVarNames.                            "accumulated classvariable names"
883: x := String sharedPools.                                 "immediate dictionaries used as shared pools"
884: x := String allSharedPools.                              "accumulated dictionaries used as shared pools"
885: x := String selectors.                                   "message selectorsfor class"
886: x := String sourceCodeAt: #size.                         "source code for specified method"
887: x := String allInstances.                                "collection of allinstances of class"
888: x := String superclass.                                  "immediate superclass"
889: x := String allSuperclasses.                             "accumulated superclasses"
890: x := String withAllSuperclasses.                         "receiver class and accumulated superclasses"
891: x := String subclasses.                                  "immediate subclasses"
892: x := String allSubclasses.                               "accumulated subclasses"
893: x := String withAllSubclasses.                           "receiver class and accumulated subclasses"
894: b := String instSize.                                    "number of named instance variables"
895: b := String isFixed.                                     "true if no indexed instance variables"
896: b := String isVariable.                                  "true if has indexed instance variables"
897: b := String isPointers.                                  "true if index instance vars contain objects"
898: b := String isBits.                                      "true if index instance vars contain bytes/words"
899: b := String isBytes.                                     "true if index instance vars contain bytes"
900: b := String isWords.                                     true if index instance vars contain words"
```

```
901: Object withAllSubclasses size.                                    "get total number of class entries"
902:
903: "============================================================================
904:  * debuging:                                                                 *
905:  ============================================================================"
906: | a b x |
907: x yourself.                                                       "returns receiver"
908: String browse.                                                    "browse specified class"
909: x inspect.                                                        "open object inspector window"
910: x confirm: 'Is this correct?'.
911: x halt.                                                          "breakpoint to open debugger window"
912: x halt: 'Halt message'.
913: x notify: 'Notify text'.
914: x error: 'Error string'.                                         "open up error window with title"
915: x doesNotUnderstand: #cmrMessage.                                "flag message is not handled"
916: x shouldNotImplement.                                            "flag message should not be implemented"
917: x subclassResponsibility.                                        "flag message as abstract"
918: x errorImproperStore.                                            "flag an improper store into indexable object"
919: x errorNonIntegerIndex.                                          "flag only integers should be used as index"
920: x errorSubscriptBounds.                                          "flag subscript out of bounds"
921: x primitiveFailed.                                               "system primitive failed"
922:
923: a := 'A1'. b := 'B2'. a become: b.                               "switch two objects"
924: Transcript show: a, b; cr.
925:
926: "============================================================================
927:  * Misc.                                                                     *
928:  ============================================================================"
929: | x |
930: "Smalltalk condenseChanges."                                     "compress the change file"
931: x := FillInTheBlank request: 'Prompt Me'.                        "prompt user for input"
932: Utilities openCommandKeyHelp
933:      ____________________________________________________________________
934:
935:
936:     [3]Chris Rathman / [4]ChrisRath@aol.com
937:
938: References
939:
940:    1. http://www.angelfire.com/tx4/cus/index.html
941:    2. mailto:ChrisRath@aol.com
942:    3. http://www.angelfire.com/tx4/cus/index.html
943:    4. mailto:ChrisRath@aol.com
```