Sets

```
#-------------------------------------------------------------------------------

# A  SET  is an unordered collection of distinct items, enclosed in '{' and '}',
# and models the mathematical notion of a finite set.
#
# Being free of the requirement to maintain the order of the items
# allows for a significantly faster implementation of set operations.


>>> s1 = set( )                          # empty set; cannot write as  { }

>>> s2 = { 1, 2, 3, 4 }

>>> s3 = { 3, 2, 4, 1 }

>>> s2 == s3                             # order does not matter
True

>>> s2 == s1
False

>>> s4 = { 1, 2, 1, 3, 2, 4 }            # duplicate items are skipped

>>> s4 == s2
True

>>> len( s1 )                            # 'len' works
0

>>> len( s2 )
4

>>> len( s4 )
4

>>> for item in s2 :                     # 'for' works, but is seldom needed
>>>     print( e )
1
2
3
4

>>> 3 in s2                              # 'in' = 'is an element of'
True

>>> 7 in s2
False

>>> { 1, 2, 3, 4 } | { 3, 4, 5, 6 }      # | : set union
{1, 2, 3, 4, 5, 6}

>>> { 1, 2, 3, 4 } & { 3, 4, 5, 6 }      # & : set intersection
{3, 4}

>>> { 1, 2, 3, 4 } - { 3, 4, 5, 6 }      # - : set difference
{1, 2}

>>> { 1, 2, 3, 4 } ^ { 3, 4, 5, 6 }      # ^ : symmetric difference
{1, 2, 5, 6}
```

```
>>> { 1, 2, 3, 4 } <= { 5, 2, 3, 1, 4 }  # <= : subset test
True

>>> { 1, 2, 3, 4 } <= { 3, 1, 4, 2 }     # <=  includes equal sets
True

>>> { 1, 2, 3, 4 } <= { 1, 3, 5, 7, 9 }
False

>>> { 1, 2, 3, 4 } < { 1, 2, 3, 4, 5 }   # < : strict subset test
True

>>> { 1, 2, 3, 4 } < { 1, 2, 3, 4 }      # <  excludes equal sets
False

#--- the operators '>=' and '>' also work, and in the expected manner

>>> s2 |= { 3, 4, 5, 6, 7, 8 }           # A |= B  abbreviates  A = A | B

>>> s2
{1, 2, 3, 4, 5, 6, 7, 8}

>>> s2 &= { 1, 3, 7, 9 }                 # A &= B  abbreviates  A = A & B

>>> s2
{1, 3, 7}

>>> s2 -= { 1, 2, 3, 4 }                 # A -= B  abbreviates  A = A - B

>>> s2
{7}

>>> s3 ^= { 3, 4, 5, 6 }                 # A ^= B  abbreviates  A = A ^ B

>>> s3
{1, 2, 5, 6}

#--- Set Comprehensions  ( note the unpredictable order of the items )

>>> { n * n for n in range( 11 ) }
{0, 1, 4, 100, 81, 64, 9, 16, 49, 25, 36}

>>> { n * n for n in range( 11 ) if n % 2 == 1 }
{1, 25, 49, 81, 9}

>>> s5 = { "abc" for n in range( 5 ) }   # a set of strings ...

>>> s5                                   # ... but no duplicate items
{'abc'}

#-------------------------------------------------------------------------------
```