Collections

```
      #   Collections are data structures
      #   A collection represents a number of individual values

      #   Simple examples of collections are sequences like: strings, lists and tuples
C001 >>>  string = 'Hello World'
C002 >>>  string
     ==>  'Hello World'
C003 >>>  list(string)  # conversion into a list of single letters
     ==>  ['H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd']

C004 >>>  mylist = ['text', 77, False, 23.7777]  # lists may contain different data types
C005 >>>  for item in mylist:   # lists (all sequences) support the for ... in ... syntax
     ...      print("item:", item, '  type:', type(item))
     p()  item: text   type: <class 'str'>
     p()  item: 77   type: <class 'int'>
     p()  item: False   type: <class 'bool'>
     p()  item: 23.7777   type: <class 'float'>
C006 >>>  77 in mylist  # sequences support the 'in' systax
     ==>  True
C007 >>>  'sense' in mylist  # 'a in b' is a boolean expression, can be True or False
     ==>  False
C008 >>>  False in mylist
     ==>  True
C009 >>>  True in mylist
     ==>  False
```

Mapping Types: Dictionaries (1)

```
      #   A dictionary is a collection, where each element has a key
C010 >>>  days = {0: 'Monday', 1: 'Tuesday', 2: 'Wednesday', 3: 'Thursday',
     ...           4: 'Friday', 5: 'Saturday', 6: 'Sunday'}
C011 >>>  days[2]   # access one element by its key
     ==>  'Wednesday'
C012 >>>  days[5]
     ==>  'Saturday'
C013 >>>  days[5] = 'Sabado'   # values can be replaced
C014 >>>  days[7] = 'Doomsday'    # new values can be added
C015 >>>  for day in range(8):
     ...      print("day[{}] = '{}'".format(day, days[day]))
     p()  day[0] = 'Monday'
     p()  day[1] = 'Tuesday'
     p()  day[2] = 'Wednesday'
     p()  day[3] = 'Thursday'
     p()  day[4] = 'Friday'
     p()  day[5] = 'Sabado'
     p()  day[6] = 'Sunday'
     p()  day[7] = 'Doomsday'

      #   keys for a dictionary can be strings (and all other static types)
C016 >>>  valid_bets = {'r': 'Rock', 'p': 'Paper', 's': 'Scissor'}
C017 >>>  user_bet = 'p'
C018 >>>  print("Your bet was '{}'".format(valid_bets[user_bet]))
     p()  Your bet was 'Paper'
```

Mapping Types: Dictionaries (2)

```
C019 >>>  empty = {}    # or
C020 >>>  empty = dict()    #  which is a bultin function
C021 >>>  newdict = dict(r='Rock', p='Paper', s='Scissor')   # another way to create a dictionary
C022 >>>  newdict   # this only works for keys, that would be valid variable names
     ==>  {'r': 'Rock', 'p': 'Paper', 's': 'Scissor'}
C023 >>>  newdict.keys()   #  dict keys are always 'unordered'
     ==>  dict_keys(['r', 'p', 's'])
C024 >>>  keylist = sorted(newdict.keys())    # return a sorted copy of the key list
C025 >>>  keylist     # this is, what we get, now use it
     ==>  ['p', 'r', 's']
C026 >>>  for short in keylist:    # print the content of a dictionary
     ...      print("key: {}, value:'{}'".format(short, newdict[short]))
     p()  key: p, value:'Paper'
     p()  key: r, value:'Rock'
     p()  key: s, value:'Scissor'
```

Mapping Types: Dictionaries (3)

```
C027 >>>  newdict = dict(r='Rock', p='Paper', s='Scissor')
       #  like the keys() method, dictionaries also have the values() and the items() methods
C028 >>>  list(newdict.values())
     ==>  ['Rock', 'Paper', 'Scissor']
C029 >>>  list(newdict.items())  # the items() returns tuples of keys + values
     ==>  [('r', 'Rock'), ('p', 'Paper'), ('s', 'Scissor')]
C030 >>>  del newdict['s']  # remove a value from a dictionary
C031 >>>  newdict
     ==>  {'r': 'Rock', 'p': 'Paper'}
```