Control Flow and reserved words

```
        #   Control Flow is about how to change the sequence of instructions

        #   Following is a list of all reserved names in Python. These names can not be used
        #   for self-defined objects
CF001 >>>   """ False      class      finally    is         return
      ...       None       continue   for        lambda     try
      ...       True       def        from       nonlocal   while
      ...       and        del        global     not        with
      ...       as         elif       if         or         yield
      ...       assert     else       import     pass
      ...       break      except     in         raise      """
        #   most of these reserved names are part of some control flow syntax
        #   The following pages will show, how the these keywords are used
```

'if' - Conditional Execution

```
CF002 >>>   boolex = True # this can be any conditional expression or boolean value
CF003 >>>   a = 'nothing'

CF004 >>>   if not boolex:
       ...      a = 'yes'   # block of statements
CF005 >>>   a    # statement was not executed
      ==>   'nothing'

CF006 >>>   if boolex:
       ...      a = 'yes'   # block of statements
CF007 >>>   a    # statement was executed
      ==>   'yes'

        #   there is an 'else' branch
CF008 >>>   if not boolex:
       ...      a = 'yes'   # block of statements
       ...  else:
       ...      a = 'no'    # alternative block of statements
CF009 >>>   a    # the else part was executed
      ==>   'no'

        #   the keywords 'if' and 'else' also occur in a 'conditional assignment'
        #   a conditional assignement allows a shortcut for the above if...else

CF010 >>>   a = 'yes' if boolex else 'no'
        #   this form of a statement is new in Python and it was introduced
        #   on a demand to have something like the C/C++ statement:
CF011 >>>   "a = boolex ? 'yes' : 'no';"
```

'if' with many conditions

```
CF012 >>>   small, medium, big = 3, 7, 15
CF013 >>>   item = 8
CF014 >>>   if item <= small:
      ...      print("small")
      ...   elif item <= medium:
      ...      print("medium")
      ...   elif item <= big:
      ...      print("big")
      ...   else:
      ...      print("huge")
      p()   big

        #   Nested 'if' statements
CF015 >>>   if True:
      ...       if True:
      ...           if False:
      ...               pass
      ...           else:
      ...               pass
      ...   else:
      ...       pass
```

'for ... in' - Process objects in a sequence

```
      #    The for statement is used to access all elements in a list
CF016 >>>  fruit = ['apple', 'pear', 'banana', 'orange', 'kiwi', 'strawberry']
CF017 >>>  for element in fruit:
      ...      print(element)
      p()  apple
      p()  pear
      p()  banana
      p()  orange
      p()  kiwi
      p()  strawberry

      #    check all elements and skip some
CF018 >>>  for fr in fruit:
      ...      if len(fr) > 5:
      ...          continue    # returns to the begin of the loop, and fetches the next element
      ...      print("selected fruit: {}".format(fr.upper()))
      p()  selected fruit: APPLE
      p()  selected fruit: PEAR
      p()  selected fruit: KIWI
CF019 >>>  fr  # the last element is still there
      ==>  'strawberry'
```

'for ... in' - just for counting

```
        #    'for' works with sequences. A sequence often is numbers from 'range()'
CF020 >>>    for num in range(3):
      ...        print("in the range:", num)
      p()    in the range: 0
      p()    in the range: 1
      p()    in the range: 2

        #    look at this:
CF021 >>>    range(3)     # this is an object, which 'generates' numbers
      ==>    range(0, 3)
CF022 >>>    list(range(3))    # This makes a list, consisting of the generated numbers
      ==>    [0, 1, 2]

        #    its good to understand the range() function
CF023 >>>    list(range(4,8))    # from 4 up to, but not including 8
      ==>    [4, 5, 6, 7]
CF024 >>>    list(range(0,10,2))   # counting in steps of 2
      ==>    [0, 2, 4, 6, 8]
CF025 >>>    list(range(13,-11,-5))    # counting can be backwards and negative
      ==>    [13, 8, 3, -2, -7]
```

Termination of a 'for' loop

```
      #    Terminate a 'for' loop at some condition (find something with 'b')
CF026 >>>  for fr in fruit:
      ...      if fr[0] == 'b':
      ...          print("element, which terminates the loop:", fr)
      ...          break
      ...      print("found:", fr)
      ...  else:
      ...      print("loop was exhausted")
      p()  found: apple
      p()  found: pear
      p()  element, which terminates the loop: banana

      #    Terminate a 'for' loop at some condition (find something with 'z')
CF027 >>>  for fr in fruit:
      ...      if fr[0] == 'z':
      ...          print("element, which terminates the loop:", fr)
      ...          break
      ...      print("found:", fr)
      ...  else:
      ...      print("nothing found with 'z'")
      p()  found: apple
      p()  found: pear
      p()  found: banana
      p()  found: orange
      p()  found: kiwi
      p()  found: strawberry
      p()  nothing found with 'z'
```

more of that

```
#   if ... elif ... else
#   for ... in ... else
#   while ... else
#   continue
#   break
#   def ... return
#   global
#   try ... except ... else ... finally
#   raise
#
#   class
#   del
#   True, False, None
#   and, or, not

#   Imperative statements are statements, which perform basic of the language
```

Imperative Statements

```
#    'import'
```