HowTo - Numeric Check

```
        #    Validating user input may turn into a tricky task, but user data should never enter
        #    unchecked. There are of course limits. Some things in this world are more complicated,
        #    than even a smart programmer might imagine. Addresses and phone numbers are good examples.
        #    Numeric values, which will be used for mathematical operations, must be checked.

        #    The easiest way is the use of int() and float()
HT1001 >>>   int(" -234 ")    # convert a string into a number (integer)
       ==>   -234
HT1002 >>>   '-56'.isnumeric()   # us the isnumeric() method on a string is not perfect
       ==>   False
HT1003 >>>   float(" 3.14159 ")   # floating points are easy to check
       ==>   3.14159
HT1004 >>>   float("-323.94e-2")   # exponential notation is considered as valid
       ==>   -3.2394
```

HowTo - Handle bad data

```
          #    Bad data leads to an exception
HT1005 >>>    int('nonumber')
       err!  ValueError("invalid literal for int() with base 10: 'nonumber'",)
          #    ... which can be handled by the program. The 'try' ... 'except' statement will be
          #    explained later. For now let's trust in Python and write a small conversion routine:

HT1006 >>>    def get_numeric(str_data):
       ...        try:
       ...            return int(str_data)
       ...        except ValueError:
       ...            return None

HT1007 >>>    print( get_numeric('33'))
       p()    33
HT1008 >>>    print( get_numeric('test'))
       p()    None
          #    the 'None' value should be tested with 'is':
HT1009 >>>    num = get_numeric('nonum')
       ...    if num is None:
       ...        print("data is not numeric, please reenter")
       p()    data is not numeric, please reenter
```

HowTo - Avoid "Magic numbers"!

```
HT1010 >>>   def RPS():
       ...       ...
       ...       scoreList=[0,0]; #PC,EU
       ...       playerList=["PC","USER"]
       ...       ...
       ...       # somewhere later in the program:
       ...       print("Result\nUser: "+str(scoreList[0])+" PC: "+str(scoreList[1]))
       ...   # What is the meaning of 0 and 1? - The mistake is easy to see here,
       ...   # but in a real program the two spots may be far from each other.

          #  Better: give numbers a name

       ...   def RPS():
       ...       USR, CMP = 0, 1    # good
       ...       USR, CMP = tuple(range(2))   # even better

       ...       players =["Comp", "User"]
       ...       scores =[0, 0]
       ...       ...
       ...       # then, later:
       ...       print("Result\n{}: {}, {}: {}".format(players[USR], scores[USR],
       ...                                             players[CMP], scores[CMP]))
```

HowTo - Check user input against valid answers

```
             #    to write a number of checks against each possible word is not really an option
HT1011 >>>   def check(input):
       ...       result = 0
       ...       if  input == 'l' or input == 'lis' or input == 'lisbon':
       ...           result = 1
       ...       elif input == 'b' or input == 'ber' or input == 'berlin':
       ...           result = 2
       ...       return result

             #    there is a method for strings, which makes things easier:
HT1012 >>>   input = 'be'
HT1013 >>>   'berlin'.startswith(input)   # this is true for 'b' or 'berl' as well
       ==>   True

             #    If there is a list of possible values, use a loop:
HT1014 >>>   def check(input, list):
       ...       for option in list:
       ...           if option.startswith(input):
       ...               return 1 + list.index(option)
       ...       else:
       ...           return 0
HT1015 >>>   input = 'Ber'
HT1016 >>>   check(input.lower(), ['lisbon', 'berlin', 'madrid', 'rome'])
       ==>   2

             #    if it is our own list of options, I would recommend the split method:
HT1017 >>>   check('ro', 'lisbon berlin madrid rome'.split())
       ==>   4
```

more about checking options

```
          #    if the list comes from an external source, we must be careful
          #    the user input could match more than on entry in the list (e.g. 'lisbon' and 'london')
          #    The check routine must return a more specific answer:
HT1018 >>>   options = "lisbon madrid berlin bern athens amsterdam".split()
HT1019 >>>   options
       ==>   ['lisbon', 'madrid', 'berlin', 'bern', 'athens', 'amsterdam']
HT1020 >>>   def check(input, olist):
       ...       opt = None
       ...       for test in olist:
       ...           if test.startswith(input.lower()):
       ...               # its a match
       ...               if opt:    # but there was another match before :-(
       ...                   return False, 'answer is too short'
       ...               else:
       ...                   opt = test   # preserve the first match
       ...       if opt:
       ...           return True, opt
       ...       return False, 'not in list'
```

Using a checking function with a detailed answer

```
            #    let's try:
HT1021 >>>  options = "lisbon madrid berlin bern rome athens amsterdam".split()
HT1022 >>>  check('z', options)
       ==>  (False, 'not in list')
HT1023 >>>  check('be', options)
       ==>  (False, 'answer is too short')
HT1024 >>>  check('berl', options)
       ==>  (True, 'berlin')

            #    or like this:
HT1025 >>>  input_list = 'r ro z l a b be berl'.split()
HT1026 >>>  for inp in input_list:
       ...      result, text = check(inp, options)
       ...      if result:
       ...          print("'{}' matches: {}".format(inp, text))
       ...      else:
       ...          print("'{}' did not work: {}".format(inp, text))
       p()  'r' matches: rome
       p()  'ro' matches: rome
       p()  'z' did not work: not in list
       p()  'l' matches: lisbon
       p()  'a' did not work: answer is too short
       p()  'b' did not work: answer is too short
       p()  'be' did not work: answer is too short
       p()  'berl' matches: berlin
```