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 1 and 2? - 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]))
```