Claudiordgz

Solutions of Data Structures and Algorithms in Python

Book Authors: Goodrich, Tamassia and Goldwasser

Contents

Python Primer

The first chapter in the book is all about learning to handle Python syntax. Subjects include objects, control flow, functions, I/O operations, exceptions, iterators and generators, namespaces, modules, and scope. There is nothing regarding python packaging to redistribute your own module, which is a subject of its own.

1.1. Format

All exercises will be presented with their own Python Doctest documentation to allow testing. To run them in your own python package you can copy paste the text and add a main like the following:

DoctestMain

```
if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

This is just to try to keep it as simple as possible while adding how to run the code in your own work environment.

Pro-Tip. JetBrains Pycharm is awesome, I really recommend it, plus they got a Community Edition if you are pennyless like me. The colors, the functionality it just rocks. **Plus the IDE can run the examples without the need of using a main function.**

Pro-Tip. I like to use Anaconda for my Python distro, but the standalone Python 2.7 or >= 3 works too.

1.1.1 Exercises

The exercises in the first chapter are fun, no joke. I've seen what's coming in chapter 2 and those exercises look terrible because they are open ended questions, but they are also important concepts.

R-1.1

Write a short Python function, `is_multiple(n, m)`, that takes two integer values and returns `True` if n is a multiple of m, that is, $n = mi$ for some integer i, and `False` otherwise.

Exercise R-1.1

```python
"""Write a short Python function, is_multiple(n, m), that takes two integer
values and returns True if n is a multiple of m, that is, n = mi for some
integer i, and False otherwise.

>>> is_multiple(50,3)
False
"""


def is_multiple(n, m):
    """Return True if n is multiple of m such that n = mi
    Else returns False

    >>> is_multiple(50,3)
    False
    >>> is_multiple(60,3)
    True
    >>> is_multiple(70,3)
    False
    >>> is_multiple(-50,2)
    True
    >>> is_multiple(-60,2)
    True
    >>> is_multiple("test",10)
    Numbers must be Integer values
    >>> is_multiple(-60,"test")
    Numbers must be Integer values
    """
    try:
        return True if (int(n) % int(m) == 0) else False
    except ValueError:
        print("Numbers must be Integer values")
```

Go to Top

R-1.2

Write a short Python function, `is_even(k)`, that takes an integer value and returns `True` if k is even, and `False` otherwise. However, your function cannot use the multiplication, modulo, or division operators.

Exercise R-1.2

```
"""Write a short Python function, is_even(k), that takes an integer value and
returns True if k is even, and False otherwise. However, your function
cannot use the multiplication, modulo, or division operators

>>> is_even(127)
False
"""



def is_even(k):
    """Return True if n is even
    Else returns False

    >>> is_even(10)
    True
    >>> is_even(9)
    False
    >>> is_even(11)
    False
    >>> is_even(13)
    False
    >>> is_even(1025)
    False
    >>> is_even("test")
    Number must be Integer values
    """
    try:
        return int(k) & 1 == 0
    except ValueError:
        print("Number must be Integer values")
```

Go to Top

R-1.3

Write a short Python function, `minmax(data)`, that takes a sequence of one or more numbers, and returns the smallest and largest numbers, in the form of a tuple of length two. Do not use the built-in functions `min` or `max` in implementing your solution.

Exercise R-1.3

```
""" Write a short Python function, minmax(data), that takes a sequence of
one or more numbers, and returns the smallest and largest numbers, in the
form of a tuple of length two. Do not use the built-in functions min or
max in implementing your solution.

>>> print(minmax([2,3,4,5,6,7,8,9,10,11,10,9,8,7,6,5,4,3,2,1]))
Min 1 - Max 11
"""

class MinMax():
    """MinMax object helper

    Attributes:
        min (int): Minimun value of attributes
        max (max): Maximum value of attributes

    """
    def __init__(self, min, max):
        """ Default Constructor

        Args:
          min (int): Number with lesser value
          max (int): Number with higher value
      """
        self.min = min
        self.max = max
    def __str__(self):
        """String representation overload
        """
        return "Min {min} - " \
               "Max {max}".format(min=str(self.min),
                                  max=str(self.max))
```

```
def minmax(data):
    """This is the algorithm to find the
    minimum and maximun in a list.

    Args:
        data (list of int): Simple array of
        Integers

    Returns:
        A tuple MinMax that holds the minimum
        and maximum values found in the list

    Examples:
        Here are some examples!

    >>> print(minmax([2,3,4,5,6,7,8,9,10,11,10,9,8,7,6,5,4,3,2,1]))
    Min 1 - Max 11
    >>> print(minmax([50,200,300,3,78,19203,56]))
    Min 3 - Max 19203
    >>> print(minmax([100,150,200,500]))
    Min 100 - Max 500
    """
    start = 0
    mm = MinMax(data[start],data[start])
    if len(data) & 1 == 1:
        if data[start] < data[start+1]:
            mm.max = data[start+1]
            mm.min = data[start]
            start += 2
        else:
            start += 1
    for index in range(start, len(data[start:]), 2):
        if data[index] < data[index+1] :
            l_min = data[index]
            l_max = data[index+1]
        else:
            l_min = data[index+1]
            l_max = data[index]
        if mm.min > l_min:
            mm.min = l_min
        if mm.max < l_max:
            mm.max = l_max
```

```
return mm
```

Go to Top

R-1.4 & R-1.5

Write a short Python function that takes a positive integer n and returns the sum of the squares of all the positive integers smaller than n.

Give a single command that computes the sum from Exercise R-1.4, relying on Python's comprehension syntax and the built-in sum function.

Exercise R-1.4 & R-1.5

```
""" Write a short Python function that takes a positive integer n and returns
the sum of the squares of all the positive integers smaller than n.

Give a single command that computes the sum from Exercise R-1.4, relying
on Pythons comprehension syntax and the built-in sum function.

>>> sum_of_squares(10)
285
"""

def sum_of_squares(n):
    """Sum of squares of postive integers
    smaller than n

    Args:
        n (int): Highest number

    >>> sum_of_squares(10)
    285
    >>> sum_of_squares(20)
    2470
    >>> sum_of_squares(500)
    41541750
    >>> sum_of_squares(37)
    16206
    >>> sum_of_squares(-1)
    False
    """
    return sum([pow(x,2) for x in range(n)]) if n > 0 else False
```

Go to Top

R-1.6 & R-1.7

Write a short Python function that takes a positive integer n and returns the sum of the squares of all the odd positive integers smaller than n.

Give a single command that computes the sum from Exercise R-1.6, relying on Python's comprehension syntax and the built-in sum function.

Exercise R-1.6 & R-1.7

```
"""Write a short Python function that takes a positive integer n and returns
the sum of the squares of all the odd positive integers smaller than n.

Give a single command that computes the sum from Exercise R-1.6, rely-
ing on Python's comprehension syntax and the built-in sum function.

"""

def sum_of_odd_squares(n):
    """Sum of squares of odd postive integers
    smaller than n

    Args:
        n (int): Highest number

    >>> sum_of_odd_squares(10)
    165
    >>> sum_of_odd_squares(20)
    1330
    >>> sum_of_odd_squares(500)
    20833250
    >>> sum_of_odd_squares(37)
    7770
    >>> sum_of_odd_squares(-1)
    False
    """
    return sum([pow(x,2) for x in range(1, n, 2)]) if n > 0 else False
```

Go to Top

R-1.8

Python allows negative integers to be used as indices into a sequence, such as a string. If string s has length n, and expression $s[k]$ is used for index $-n \leq k < 0$, what is the equivalent index $j \geq 0$ such that $s[j]$ references the same element?

Exercise R-1.8

```
"""Python allows negative integers to be used as indices into a sequence,
such as a string. If string s has length n, and expression s[k] is used for in-
dex -n<=k<0, what is the equivalent index j>=0such that s[j] references
the same element?

>>> l = [2,3,4,5,6,7,8,9,10,11,10,9,8,7,6,5,4,3,2,1]
>>> return_element(l, 0)
(2, -20)
>>> return_element(l, 1)
(3, -19)
>>> return_element(l, 2)
(4, -18)
"""



def return_element(data, k):
    """Tells you the equivalent negative index

    Args:
        data (list of int): Simple array
        k (int): index you want to know
        the equivalent negative index

    Returns:
        (val, index)
        val (object): element at position k
        index: negative index of that position
    Examples:
        Here are some examples!

    >>> l = [2,3,4,5,6,7,8,9,10,11,10,9,8,7,6,5,4,3,2,1]
    >>> return_element(l, 0)
```

```
(2, -20)
>>> return_element(l, 1)
(3, -19)
>>> return_element(l, 2)
(4, -18)
"""
idx = k-len(data)
return data[idx], idx if data else False
```

Go to Top

R-1.9

What parameters should be sent to the range constructor, to produce a range with values 50, 60, 70, 80?

Exercise R-1.9

```
"""What parameters should be sent to the range constructor, to produce a
range with values 50, 60, 70, 80?

>>> range_from_fifty()
[50, 60, 70, 80]
"""

def range_from_fifty():
    """ Creates a list
    with values 50, 60, 70, 80

     Returns:
        list: [50, 60, 70, 80]

    >>> range_from_fifty()
    [50, 60, 70, 80]
    """
    return range(50,81,10)
```

Go to Top

R-1.10

What parameters should be sent to the range constructor, to produce a range with values 8, 6, 4, 2, 0, -2, -4, -6, -8?

Exercise R-1.10

```
""" What parameters should be sent to the range constructor, to produce a
range with values 8, 6, 4, 2, 0, -2, -4, -6, -8?

>>> range_from_eigth()
[8, 6, 4, 2, 0, -2, -4, -6, -8]
"""

def range_from_eigth():
    """ Return the list [8, 6, 4, 2, 0, -2, -4, -6, -8]
    :return:
        the list [8, 6, 4, 2, 0, -2, -4, -6, -8]
    >>> range_from_eigth()
    [8, 6, 4, 2, 0, -2, -4, -6, -8]
    """
    return range(8, -9, -2)
```

Go to Top

R-1.11

Demonstrate how to use Python's list comprehension syntax to produce the list [1, 2, 4, 8, 16, 32, 64, 128, 256].

Exercise R-1.11

```
"""Demonstrate how to use Python's list
comprehension syntax to produce the list
[1, 2, 4, 8, 16, 32, 64, 128, 256].

>>> list_comprehension_example()
[1, 2, 4, 8, 16, 32, 64, 128, 256]
"""
def list_comprehension_example():
    """ Return list
    [1, 2, 4, 8, 16, 32, 64, 128, 256]

    :return:
        list: [1, 2, 4, 8, 16, 32, 64, 128, 256]

    >>> list_comprehension_example()
    [1, 2, 4, 8, 16, 32, 64, 128, 256]
    """
    return [pow(2,x) for x in range(9)]
```

Go to Top

R-1.12

This is a random method, I usually stress test anything that is random since I am always uneasy about it.

Python's random module includes a function choice(data) that returns a random element from a non-empty sequence. The random module includes a more basic function randrange, with parametrization similar to the built-in range function, that return a random choice from the given range. Using only the randrange function, implement your own version of the choice function.

Exercise R-1.12

```
""" Python's random module includes a function choice(data) that returns a
random element from a non-empty sequence. The random module includes
a more basic function randrange, with parametrization similar to
the built-in range function, that return a random choice from the given
range. Using only the randrange function, implement your own version
of the choice function.

>>> data = [2,3,4,5,6,7,8,9,10,11,10,9,8,7,6,5,4,3,2,1]
>>> results = list()
>>> for x in range(len(data)*20):
...     val = custom_choice(data)
...     results.append(val in data)
>>> print(results)
[True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
```

```
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True, \
True, True, True, True, True, True, True, True, True, True]
"""

def custom_choice(data):
    import random
    return data[random.randrange(0,len(data))]
```

Go to Top

C-1.13

Write a pseudo-code description of a function that reverses a list of n integers, so that the numbers are listed in the opposite order than they were before, and compare this method to an equivalent Python function for doing the same thing.

Exercise C-1.13

```
"""Write a pseudo-code description of a function that reverses a list of n
integers, so that the numbers are listed in the opposite order than they
were before, and compare this method to an equivalent Python function
for doing the same thing.


>>> l1 = [2,3,4,5,6,7,8,9,10,11,10,9,8,7,6,5,4,3,2,1]
>>> custom_reverse(l1)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2]
"""
import cProfile

def custom_reverse(data):
    """ Reverse the data array

    :param data: a list of elements
    :return: reverse list

    >>> l1 = [2,3,4,5,6,7,8,9,10,11,10,9,8,7,6,5,4,3,2,1]
    >>> custom_reverse(l1)
    [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2]
    """
    return [data[len(data)-x-1] for x in range(len(data))]

def standard_reverse(data):
    return reversed(data)

def other_reverse(data):
    return data[::-1]

if __name__ == "__main__":
    l1 = [2,3,4,5,6,7,8,9,10,11,10,9,8,7,6,5,4,3,2,1]
```

```
l2 = [2,3,4,5,6,7,8,9,10,11,10,9,8,7,6,5,4,3,2,1]
l3 = [2,3,4,5,6,7,8,9,10,11,10,9,8,7,6,5,4,3,2,1]
cProfile.run('custom_reverse(l1)')
cProfile.run('standard_reverse(l2)')
cProfile.run('other_reverse(l3)')
```

cProfile Results

Here is a simple cProfile with the results. Time shows as 0.000 but the number of function calls tell us our implementation is not that good.

`25 function calls in 0.000 seconds`

`Ordered by: custom_reverse`

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
1	0.000	0.000	0.000	0.000	<string>:1(<module>)
1	0.000	0.000	0.000	0.000	c113.py:14(custom_reverse)
21	0.000	0.000	0.000	0.000	len
1	0.000	0.000	0.000	0.000	method 'disable' of '_lsprof.Profiler' objects
1	0.000	0.000	0.000	0.000	range

`3 function calls in 0.000 seconds`

`Ordered by: standard_reverse`

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
1	0.000	0.000	0.000	0.000	<string>:1(<module>)
1	0.000	0.000	0.000	0.000	c113.py:26(standard_reverse)
1	0.000	0.000	0.000	0.000	method 'disable' of '_lsprof.Profiler' objects

`3 function calls in 0.000 seconds`

`Ordered by: other_reverse`

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
1	0.000	0.000	0.000	0.000	<string>:1(<module>)
1	0.000	0.000	0.000	0.000	c113.py:29(other_reverse)
1	0.000	0.000	0.000	0.000	method 'disable' of '_lsprof.Profiler' objects

Go to Top

C-1.14

Write a short Python function that takes a sequence of integer values and determines if there is a distinct pair of numbers in the sequence whose product is odd.

In this method what we do first is remove all the repeated elements, then we just extract all the odd numbers. Multiply any of the odd numbers and you get an odd number. Any even number multiplied by an odd number returns an even number.

Exercise C-1.14

```
""" Write a short Python function that takes a sequence of integer values and
determines if there is a distinct pair of numbers in the sequence whose
product is odd.

>>> data = [2,3,4,5,6,7,8,9,10,11,10,9,8,7,6,5,4,3,2,1]
>>> get_odd_numbers(data)
[1, 3, 5, 7, 9, 11]
"""

def get_odd_numbers(data):
    """ Returns all the numbers whose product is odd

    :param data: A list of integers
    :return: numbers whose product is odd
    """
    s = set(data)
    return [k for k in s if (k & 1 == 1)]
```

Go to Top

C-1.15

Write a Python function that takes a sequence of numbers and determines if all the numbers are different from each other (that is, they are distinct).

We just build a set from the original list, if the length of the set is smaller, we have repeated numbers. Besides this, we could also sort the elements and look for a repeated element.

Exercise C-1.15

```python
""" Write a Python function that takes a sequence of numbers and determines
if all the numbers are different from each other (that is, they are distinct).

>>> data = [2,3,4,5,6,7,8,9,10,11,10,9,8,7,6,5,4,3,2,1]
>>> check_if_unique(data)
False
>>> check_if_unique(data[:9])
True
"""

def check_if_unique(data):
    """ Check if all elements in list are unique

    :param data: list of integers
    :return: True if all unique, else False
    """
    s = set(data)
    return len(s) == len(data)
```

Go to Top

C-1.16 & C-1.17

In our implementation of the scale function (page 25), the body of the loop executes the command `data[j] *= factor`. We have discussed that numeric types are immutable, and that use of the *= operator in this context causes the creation of a new instance (not the mutation of an existing instance). How is it still possible, then, that our implementation of scale changes the actual parameter sent by the caller?

Had we implemented the scale function (page 25) as follows, does it work properly?

The incorrect_way

```
def scale(data, factor):
  for val in data:
    val *= factor
```

Explain why or why not.

Well... here are some inmutable types in Python.

- numerical types
- string types
- types

And here are some mutable types.

- lists
- dicts
- classes

So taken by that premise, if we want a function that modifies numeric values in place, we would need a `class` that wraps that numeric value. This is not trivial. So we are going to take respectfully a class done by The Edwards Research group (http://www.edwards-research.com/), this class is presented here: http://blog.edwards-research.com/2013/09/mutable-numeric-types-in-python/.

It is everything you dreamed of, I give you, a mutable numeric.

Mutable Numeric

```
"""
MutableNum class from: http://blog.edwards-research.com/2013/09/mutable-numeric-types-in-python/

Allows you to pass the instance to a function, and with proper coding, allows you to modify the
value of the instance inside the function and have the modifications persist.

For example, consider:

>     def foo(x): x *= 2
>     x = 5
>     foo(x)
>     print(x)

This will print 5, not 10 like you may have hoped.  Now using the MutableNum class:

>     def foo(x): x *= 2
>     x = MutableNum(5)
>     foo(x)
>     print(x)

This *will* print 10, as the modifications you made to x inside of the function foo will persist.

Note, however, that the following *will not* work:

>     def bar(x): x = x * 2
>     x = MutableNum(5)
>     bar(x)
>     print(x)

The difference being that [x *= 2] modifies the current variable x, while [x = x * 2] creates a new
variable x and assigns the result of the multiplication to it.

If, for some reason you can't use the compound operators ( +=, -=, *=, etc.), you can do something
like the following:

>     def better(x):
```

```
>         t = x
>         t = t * 2
>         # ... (Some operations on t) ...
>
>         # End your function with a call to x.set()
>         x.set(t)

"""
class MutableNum(object):
    __val__ = None
    def __init__(self, v): self.__val__ = v
    # Comparison Methods
    def __eq__(self, x):        return self.__val__ == x
    def __ne__(self, x):        return self.__val__ != x
    def __lt__(self, x):        return self.__val__ <  x
    def __gt__(self, x):        return self.__val__ >  x
    def __le__(self, x):        return self.__val__ <= x
    def __ge__(self, x):        return self.__val__ >= x
    def __cmp__(self, x):       return 0 if self.__val__ == x else 1 if self.__val__ > 0 else -1
    # Unary Ops
    def __pos__(self):          return self.__class__(+self.__val__)
    def __neg__(self):          return self.__class__(-self.__val__)
    def __abs__(self):          return self.__class__(abs(self.__val__))
    # Bitwise Unary Ops
    def __invert__(self):       return self.__class__(~self.__val__)
    # Arithmetic Binary Ops
    def __add__(self, x):       return self.__class__(self.__val__ + x)
    def __sub__(self, x):       return self.__class__(self.__val__ - x)
    def __mul__(self, x):       return self.__class__(self.__val__ * x)
    def __div__(self, x):       return self.__class__(self.__val__ / x)
    def __mod__(self, x):       return self.__class__(self.__val__ % x)
    def __pow__(self, x):       return self.__class__(self.__val__ ** x)
    def __floordiv__(self, x):  return self.__class__(self.__val__ // x)
    def __divmod__(self, x):    return self.__class__(divmod(self.__val__, x))
    def __truediv__(self, x):   return self.__class__(self.__val__.__truediv__(x))
    # Reflected Arithmetic Binary Ops
    def __radd__(self, x):      return self.__class__(x + self.__val__)
    def __rsub__(self, x):      return self.__class__(x - self.__val__)
    def __rmul__(self, x):      return self.__class__(x * self.__val__)
    def __rdiv__(self, x):      return self.__class__(x / self.__val__)
    def __rmod__(self, x):      return self.__class__(x % self.__val__)
    def __rpow__(self, x):      return self.__class__(x ** self.__val__)
    def __rfloordiv__(self, x): return self.__class__(x // self.__val__)
```

```
def __rdivmod__(self, x):    return self.__class__(divmod(x, self.__val__))
def __rtruediv__(self, x):   return self.__class__(x.__truediv__(self.__val__))
# Bitwise Binary Ops
def __and__(self, x):        return self.__class__(self.__val__ & x)
def __or__(self, x):         return self.__class__(self.__val__ | x)
def __xor__(self, x):        return self.__class__(self.__val__ ^ x)
def __lshift__(self, x):     return self.__class__(self.__val__ << x)
def __rshift__(self, x):     return self.__class__(self.__val__ >> x)
# Reflected Bitwise Binary Ops
def __rand__(self, x):       return self.__class__(x & self.__val__)
def __ror__(self, x):        return self.__class__(x | self.__val__)
def __rxor__(self, x):       return self.__class__(x ^ self.__val__)
def __rlshift__(self, x):    return self.__class__(x << self.__val__)
def __rrshift__(self, x):    return self.__class__(x >> self.__val__)
# Compound Assignment
def __iadd__(self, x):       self.__val__ += x; return self
def __isub__(self, x):       self.__val__ -= x; return self
def __imul__(self, x):       self.__val__ *= x; return self
def __idiv__(self, x):       self.__val__ /= x; return self
def __imod__(self, x):       self.__val__ %= x; return self
def __ipow__(self, x):       self.__val__ **= x; return self
# Casts
def __nonzero__(self):       return self.__val__ != 0
def __int__(self):           return self.__val__.__int__()              # XXX
def __float__(self):         return self.__val__.__float__()            # XXX
def __long__(self):          return self.__val__.__long__()             # XXX
# Conversions
def __oct__(self):           return self.__val__.__oct__()              # XXX
def __hex__(self):           return self.__val__.__hex__()              # XXX
def __str__(self):           return self.__val__.__str__()              # XXX
# Random Ops
def __index__(self):         return self.__val__.__index__()            # XXX
def __trunc__(self):         return self.__val__.__trunc__()            # XXX
def __coerce__(self, x):     return self.__val__.__coerce__(x)
# Represenation
def __repr__(self):          return "%s(%d)" % (self.__class__.__name__, self.__val__)
# Define innertype, a function that returns the type of the inner value self.__val__
def innertype(self):         return type(self.__val__)
# Define set, a function that you can use to set the value of the instance
def set(self, x):
    if   isinstance(x, (int, long, float)): self.__val__ = x
    elif isinstance(x, self.__class__): self.__val__ = x.__val__
    else: raise TypeError("expected a numeric type")
```

```
    # Pass anything else along to self.__val__
    def __getattr__(self, attr):
        print("getattr: " + attr)
        return getattr(self.__val__, attr)
```

Now onto the second question... if numeric types are not valid then of course that method is not going to work. You would need to do something **radical** like returning a **new list**. As follows:

Returning new_list

```
def scale(data, factor):
    return [x*factor for x in data]
```

We can also change the list number by replacing the value in the list, remember that lists are mutable, so we can change that, we just can't change the number.

Exercise C-1.16 & C-1.17

```
""" In our implementation of the scale function (page 25), the body of the loop
executes the command data[j] *= factor. We have discussed that numeric
types are immutable, and that use of the *= operator in this context causes
the creation of a new instance (not the mutation of an existing instance).
How is it still possible, then, that our implementation of scale changes the
actual parameter sent by the caller?

Had we implemented the scale function (page 25) as follows, does it work
properly?
def scale(data, factor):
    for val in data:
        val *= factor

Explain why or why not.

>>> import MutableNum
>>> l = [2,3,4,5,6]
>>> e = [MutableNum.MutableNum(x) for x in l]
>>> scale(e, 5)
[MutableNum(10), MutableNum(15), MutableNum(20), MutableNum(25), MutableNum(30)]
>>> scale_in_place(e,0.2)
>>> print(e)
[MutableNum(2), MutableNum(3), MutableNum(4), MutableNum(5), MutableNum(6)]
"""
```

```
def scale_in_place(data, factor):
    """scales list to factor in place

    :param data: the input list of numbers
    :param factor: product factor
    :return: list with modified values
    """
    for j in range(len(data)):
        data[j] *= factor

def scale(data, factor):
    """scales list to factor

    :param data: the input list of numbers
    :param factor: product factor
    :return: list with modified values
    """
    for j in range(len(data)):
        data[j] *= factor
    return data
```

Object-Oriented Programming

The second chapter looks to provide a generic foundation on Object Orientation.

2.0.2 Exercises

The exercises in the first chapter mostly open questions.