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

Go to Top

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

```
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")
```

R-1.2

Go to Top

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

```
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("Numbers must be Integer values")
```

R-1.3

Go to Top

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

```
class MinMax():
    """MinMax object helper

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

    """
    def __init__(self, min, max):
        """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
```

R-1.4 & R-1.5

Go to Top

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

```
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
```

R-1.6 & R-1.7

Go to Top

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

```
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
```

R-1.8

Go to Top

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

```
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
```

R-1.9

Go to Top

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

Exercise R-1.9

```
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)
```

R-1.10

Go to Top

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

```
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)
```

R-1.11

Go to Top

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

```
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)]
```

R-1.12

Go to Top

This is a random method, I usually stress test anything that is random since I am always uneasy about it. Because the test is so long I placed it separately.

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

```
def custom_choice(data):
    """ Returns random element from a non-empty sequence

    :returns:
        integer value of list
    """
    import random
    return data[random.randrange(0,len(data))]
```

Pro-Tip. Stress test your random functions

Hereafter you can see the Doctest for this exercise. I separated it because it is hude. In this case we are testing a random method that is part of a library, which is overkill. But the test is not really hard to make or design so it is not much of an investment in either time or money.

Exercise R-1.12 Doctest

```
""" 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]
"""
```