Claudiordgz

Solutions of Data Structures and Algorithms in Python

Book Authors: Goodrich, Tamassia and Goldwasser

Contents

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:

Running Doctest

```
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.

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.

2.0.1 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.

Go to Top

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

```
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

```
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")
```

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

```
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
```

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

```
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

```
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
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
```