Contents

1 Basic Test Results

```
Starting tests...
Thu Nov 14 17:56:04 IST 2013
4dc1c60f593111c1042d17a0ebdd304710f117f2  -


battleship.py
ex4.py
README

Testing README...
Done testing README...

Testing ex4.py...
result_code    bubble    20    1
result_code    badinputs    20    1
result_code    choose    20    1
result_code    opponent    20    1
result_code    king    25    1
result_code    keygetter    10    1
Done testing ex4.py

Testing battleship.py...
result_code    boards    13    1
result_code    badboards    6    1
result_code    firemiss    113    1
result_code    firehit    16    1
result_code    fireillegal    47    1
result_code    place    973    1
Done testing battleship.py

Grading summary
---------------
****** king:
Number of failed tests: 0
Total number of tests : 25
Penalty: 0.0
****** bubble:
Number of failed tests: 0
Total number of tests : 20
Penalty: 0.0
****** choose:
Number of failed tests: 0
Total number of tests : 20
Penalty: 0.0
****** keygetter:
Number of failed tests: 0
Total number of tests : 10
Penalty: 0.0
****** opponent:
Number of failed tests: 0
Total number of tests : 20
Penalty: 0.0
****** badinputs:
Number of failed tests: 0
Total number of tests : 20
Penalty: 0.0
****** boards:
Number of failed tests: 0
Total number of tests : 13
```

```
Penalty: 0.0
****** badboards:
Number of failed tests: 0
Total number of tests : 6
Penalty: -0.0
****** place:
Number of failed tests: 0
Total number of tests : 973
Penalty: 0.0
****** firehit:
Number of failed tests: 0
Total number of tests : 16
Penalty: 0.0
****** firemiss:
Number of failed tests: 0
Total number of tests : 113
Penalty: 0.0
****** fireillegal:
Number of failed tests: 0
Total number of tests : 47
Penalty: 0.0
******
Expected automatic grade: 100.0
******
Submission passed!
Tests completed
```

2 aaa expected autograde

```
Grading summary
---------------
****** king:
Number of failed tests: 0
Total number of tests : 25
Penalty: 0.0
****** bubble:
Number of failed tests: 0
Total number of tests : 20
Penalty: 0.0
****** choose:
Number of failed tests: 0
Total number of tests : 20
Penalty: 0.0
****** keygetter:
Number of failed tests: 0
Total number of tests : 10
Penalty: 0.0
****** opponent:
Number of failed tests: 0
Total number of tests : 20
Penalty: 0.0
****** badinputs:
Number of failed tests: 0
Total number of tests : 20
Penalty: 0.0
****** boards:
Number of failed tests: 0
Total number of tests : 13
Penalty: 0.0
****** badboards:
Number of failed tests: 0
Total number of tests : 6
Penalty: -0.0
****** place:
Number of failed tests: 0
Total number of tests : 973
Penalty: 0.0
****** firehit:
Number of failed tests: 0
Total number of tests : 16
Penalty: 0.0
****** firemiss:
Number of failed tests: 0
Total number of tests : 113
Penalty: 0.0
****** fireillegal:
Number of failed tests: 0
Total number of tests : 47
Penalty: 0.0
******
Expected automatic grade: 100.0
******
Submission passed!
```

3 aaa hint result.png

4 README

```
USER: borgr
ID: 305385338
leshem choshen

I did not ask anyone about this excrecise.

Notice that the sequence
goes like that:
for n=1 p=pension start x = expenses
 an = growths in percentage
p*a1=X
for n=2
p*a1a2/(a2+1) = x
p*a1a2a3/((a2+1)a3)+1=x
and so on
therefore there is no use for epsilon.

==================
    Description
==================
In this you have everything for
the pensioner.
a game of battleships, and functions
helping to use your pension for renting.
Renting the most valueable house or
best one according to value
and game opponents.

==================
    Containing
==================
ex4.py A file containing different
 functions for pensions' calculations
and battleships game.
battleship.py battlships functions

==================
      Usage
==================
python3 ex4.py
python3 battleship.py
```

5 battleship.py

```
1  ##################################################################
2  # FILE: battleship.py
3  # WRITER : Leshem Choshen + borgr + 305385338
4  # EXERCISE : intro2cs ex4 200132014
5  # Description: Battleshipe game
6  ##################################################################
7
8  """Implement the following function according the description in ex4"""
9
10 def print_board(board):
11     for ind in board:
12         print(ind)
13
14 def new_board(width=10,height=None):
15     """creates a new board game for a Battleship game.
16
17     Args:
18     -width: a positive int - the width of the board - default value 10
19     -height: a positive int - the height of the board - if not spcified
20     should be as width
21
22     return: a NEW enpty board - each inner arrays is a list of 'None's.
23
24     n case of bad input: values are out of range returns None
25
26     You can assume that the types of the input arguments are correct."""
27     if height == None:
28         height = width
29     if height <= 0 or width <= 0:
30         return
31     ls=[]
32     for ind in range(height):
33         ls.append(list(None for leng in range(width)))
34     return ls
35
36
37
38 def place_ship(board,ship_length,bow,ship_direction):
39     """Put a new ship on the board
40
41     put a new ship (with unique index) on the board.
42     in case of successful placing edit the board according to the definitions
43     in the ex description.
44
45     Args:
46     -board - battleshipe board - you can assume its legal
47     -ship_length: a positive int the length of the ship
48     -bow: a tuple of ints the index of the ship's bow
49     -ship_direction: a tuple of ints representing the direction the ship
50     is facing (dx,dy) - should be out of the 4 options(E,N,W,S):
51     (1,0) -facing east, rest of ship is to west of bow,
52     (0,-1) - facing north, rest of ship is to south of bow, and etc.
53
54     return: the index of the placed ship, if the placement was successful,
55     and 'None' otherwise.
56
57     In case of bad input: values are out of range returns None
58
59      You can assume the board is legal. You can assume the other inputs
```

```
60          are of the right form. You need to check that they are legal."""
61
62      #Check if input is right
63      if (ship_direction[0]*ship_direction[1] != 0 or
64          abs(ship_direction[0]+ship_direction[1]) != 1 or
65          ship_length < 1 or bow[0] < 0 or bow[1] < 0):
66          return
67
68      #Check if there is no ship and not out of bounds
69      for ind in range(ship_length):
70          x = bow[0] - ship_direction[0]*ind
71          y = bow[1] - ship_direction[1]*ind
72          if (len(board) <= y  or len(board[0]) <= x or
73              x+y < abs(x)+abs(y) or board[y][x] != None):
74              return
75
76      #looks for highest index
77      indx = 0
78      for y in board:
79          for x in y:
80              if x != None and x[0] > indx:
81                  indx = x[0]
82
83      #places the ship
84      indx += 1
85      run = ship_length-1 #index to run through ####why notwithfor
86      list_length = [ship_length]
87      while run != -1:
88          x = bow[0] - ship_direction[0]*run
89          y = bow[1] - ship_direction[1]*run
90          board[y][x] = (indx, run, list_length)
91          run -= 1
92      return indx
93
94
95  def fire(board,target):
96      """implement a fire in battleship game
97
98      Calling this function will try to destroy a part in one of the ships on the
99      board. In case of successful fire destroy the relevant part
100     in the damaged ship by deleting it from the board. deal also with the case
101     of a ship which was completely destroyed
102
103     -board - battleshipe board - you can assume its legal
104     -target: a tuple of ints (x,y) indices on the board
105     in case of illegal target return None
106
107     returns: a tuple (hit,ship), where hit is True/False depending if the the
108     shot hit, and ship is the index of the ship which was completely
109     destroyed, or 0 if no ship was completely destroyed. or 0 if no ship
110     was completely destroyed.
111
112     Return None in case of bad input
113
114     You can assume the board is legal. You can assume the other inputs
115     are of the right form. You need to check that they are legal."""
116     #ready - start variables
117     x = target[0]
118     y = target[1]
119
120     #aim - check where you fire
121     if (len(board) <= y or len(board[0]) <= x or
122         x < 0 or y < 0):
123         return
124
125     #fire!
126     if board[y][x] == None:
127         return False, 0
```

```
128         board[y][x][2][0] -= 1
129         indx = board[y][x][0]
130         if board[y][x][2][0] == 0:
131             board[y][x] = None
132             return True, indx
133         board[y][x] = None
134         return True, 0
```

6 ex4.py

```
1  ##############################################################
2  # FILE : ex4.py
3  # WRITER : Leshem Choshen + borgr + 305385338
4  # EXERCISE : intro2cs ex4 200132014
5  # DESCRIPTION: functions for post retirement
6  #
7  ##############################################################
8
9
10 def variable_pension(salary, save, growth_rates):
11     """ calculate retirement fund assuming variable_pension
12
13     A function that calculates the value of a retirement fund in each year
14     based on the worker salary, savings,  and a list of growthRates values.
15     Number of working years is as the length of growthRats
16
17     Args:
18     - salary: the amount of money you earn each year, a non negative float.
19     - save: the percent of your salary to save in the investment account
20     each working year -  a non negative float between 0 and 100
21     - growth_rates: the annual percent increase/decrease in your investment
22     account - a float larger than or equal to -100
23
24     return: a list whose values are the size of your retirement account at
25     the end of each year.
26
27     In case of bad input: values are out of range
28     returns None
29
30     You can assume that the types of the input arguments are correct. """
31     perc = 0.01 #percent
32     if len(growth_rates) > 0 and growth_rates[0] < -100:
33         return
34     if salary >= 0 and save >= 0 and save < 101 and len(growth_rates) >= 1:
35         rtrn = [salary*save*perc]
36         for ind in range(len(growth_rates)-1):
37             if growth_rates[ind+1] < -100:
38                 return
39             rtrn.append(rtrn[ind]*(1 + growth_rates[ind+1]*perc) +
40                         salary*save*perc)
41     elif len(growth_rates) == 0:
42         rtrn = []
43     else:
44         return
45     return rtrn
46
47
48
49
50 def live_like_a_king(salary, save, pre_retire_growth_rates,
51                      post_retire_growth_rates, epsilon):
52
53     """ Find the maximal expenses you may expend during your lifetime
54
55     A function that calculates what is the maximal annual expenses you may
56     expend each year and not enter into debts
57     You may Calculate it using binary search or using arithmetics
58     Specify in your README in which method you've implemnted the function
59
```

```
    Args:
    -salary: the amount of money you make each year-a non negative float.
    -save: the percent of your salary to save in the investment account
    each working year -  a non negative float between 0 and 100
    -pre_retire_growth_rates: a list of annual growth percentages in your
    investment account - a list of floats larger than or equal to -100.
    -post_retire_growth_rates: a list of annual growth percentages
    on investments while you are retired. a list of floats larger
    than or equal to -100. In case of empty list return None
    - epsilon: an upper bound on the money must remain in the account
    on the last year of retirement. A float larger than 0

    Returns the maximal expenses value you found (such that the amount of
    money left in your account will be positive but smaller than epsilon)

    In case of bad input: values are out of range returns None

    You can assume that the types of the input arguments are correct."""
    perc = 0.01 #percent

    #check input
    if not post_retire_growth_rates or epsilon <= 0:
        return
    product = variable_pension(salary, save, pre_retire_growth_rates)
    if product == None:
        return
    if len(product) == 0:
        return 0
    else:
        product = product[-1]
    denominator = 0
    for ind in post_retire_growth_rates:
        if ind < -100:
            return

    #calculate
    for ind in post_retire_growth_rates:
        product = product*(1 + ind*perc)
        denominator = denominator*(1 + ind*perc) +1

    return product/denominator




def bubble_sort_2nd_value(tuple_list):
    """sort a list of tuples using bubble sort algorithm

    Args:
    tuples_list - a list of tuples, where each tuple is composed of a string
    value and a float value - ('house_1',103.4)

    Return: a NEW list that is sorted by the 2nd value of the tuple,
    the numerical one. The sorting direction should be from the lowest to the
    largest. sort should be stable (if values are equal, use original order)

    You can assume that the input is correct."""
    ordered = tuple_list.copy()
    for outr in range(len(ordered)):
        for inr in range(len(ordered)-outr-1):
            if(ordered[inr][1] > ordered[inr+1][1]):
                ordered[inr+1],ordered[inr] = ordered[inr],ordered[inr+1]
    return ordered
```

```python
def choosing_retirement_home(savings,growth_rates,retirement_houses):
    """Find the most expensive retirement house one can afford.

    Find the most expensive, but affordable, retiremnt house.
    Implemnt the function using binary search

    Args:
    -savings: the initial amount of money in your savings account.
    -growth_rates: a list of annual growth percentages in your
    investment account - a list of floats larger than or equal to -100.
    -retirement_houses: a list of tuples of retirement_houses, where
    the first value is a string - the name of the house and the
    second is the annual rent of it - nonnegative float.

    Return: a string - the name of the chosen retirement house
    Return None if can't afford any house.

    You need to test the legality of savings and growth_rates
    but you can assume legal retirement_house list
    You can assume that the types of the input are correct"""

    savings = live_like_a_king(savings, 100, [0], growth_rates, 0.1)

    #Checks special cases
    if savings == None:
        return
    if not retirement_houses:
        return

    # Define the variables beggining point
    retirement_houses = bubble_sort_2nd_value(retirement_houses)
    minus_infinity= -float("inf")
    afford = minus_infinity # a smaller number than possible otherwise
    mn = 0
    mx = len(retirement_houses)

    # search
    while mn<mx:
        mdl = (mn+mx)//2
        mdl_val = retirement_houses[mdl][1]
        if savings > mdl_val:
            mn = mdl+1
            afford = retirement_houses[mdl][0]
        elif savings < retirement_houses[mdl][1]:
            mx = mdl
        else:
            afford = retirement_houses[mdl][0]
            return afford

    #if no house was affordable
    if afford == minus_infinity:
        return None
    return afford

def get_value_key(value = 0):
    """returns a function that calculates the new value of a house


    #Args:
    -value: the value added per opponent - a float - the default value is 0

    This function returns a function that accepts triple containing
    (house ,annual rent,number of opponents) and returns the new value of
    this house - annual_rent+value*opponents

    You can assume that the input is correct."""

    return lambda triple: triple[1]+value*triple[2]
```

```
196
197
198
199
200
201 def choose_retirement_home_opponents(budget,key,retirement_houses):
202     """ Find the best retiremnt house that is affordable and fun
203
204     A function that returns the best retiremnt house to live in such that:
205     the house is affordable and
206     his value (annual_rent+value*opponents) is the highest
207
208     Args:
209     -annual_budget: positive float. The amount of money you can
210     expand per year.
211     -key: a function of the type returned by get_value_key
212     -retirement_houses: a list of houses (tuples), where  the first value
213     is a string - the name of the house,
214     the second is the annual rent on it - a non negative float, and the third
215     is the number of battleship opponents the home hosts - non negative int
216
217     Returns the name of the retirement home which provides the best value and
218     which is affordable.
219
220     You need to test the legality of annual_budget,
221     but you can assume legal retirement_house list
222     You can assume that the types of the input are correct"""
223     if budget <= 0:
224         return
225     minus_infinity = -float("inf")
226     mx = minus_infinity #to make sure something was added
227     for ind in retirement_houses:
228         crnt = key(ind)
229         if mx < crnt and ind[1] <= budget:
230             mx = crnt
231             name = ind[0]
232     if mx == minus_infinity:
233         return
234     return name
235
236
237
238
```