# Contents


1 Basic Test Results 2

2 aaa expected autograde 4

3 aaa hint result.png 5

4 README 6

5 battleship.py 7

6 ex4.py 10

# 1 Basic Test Results

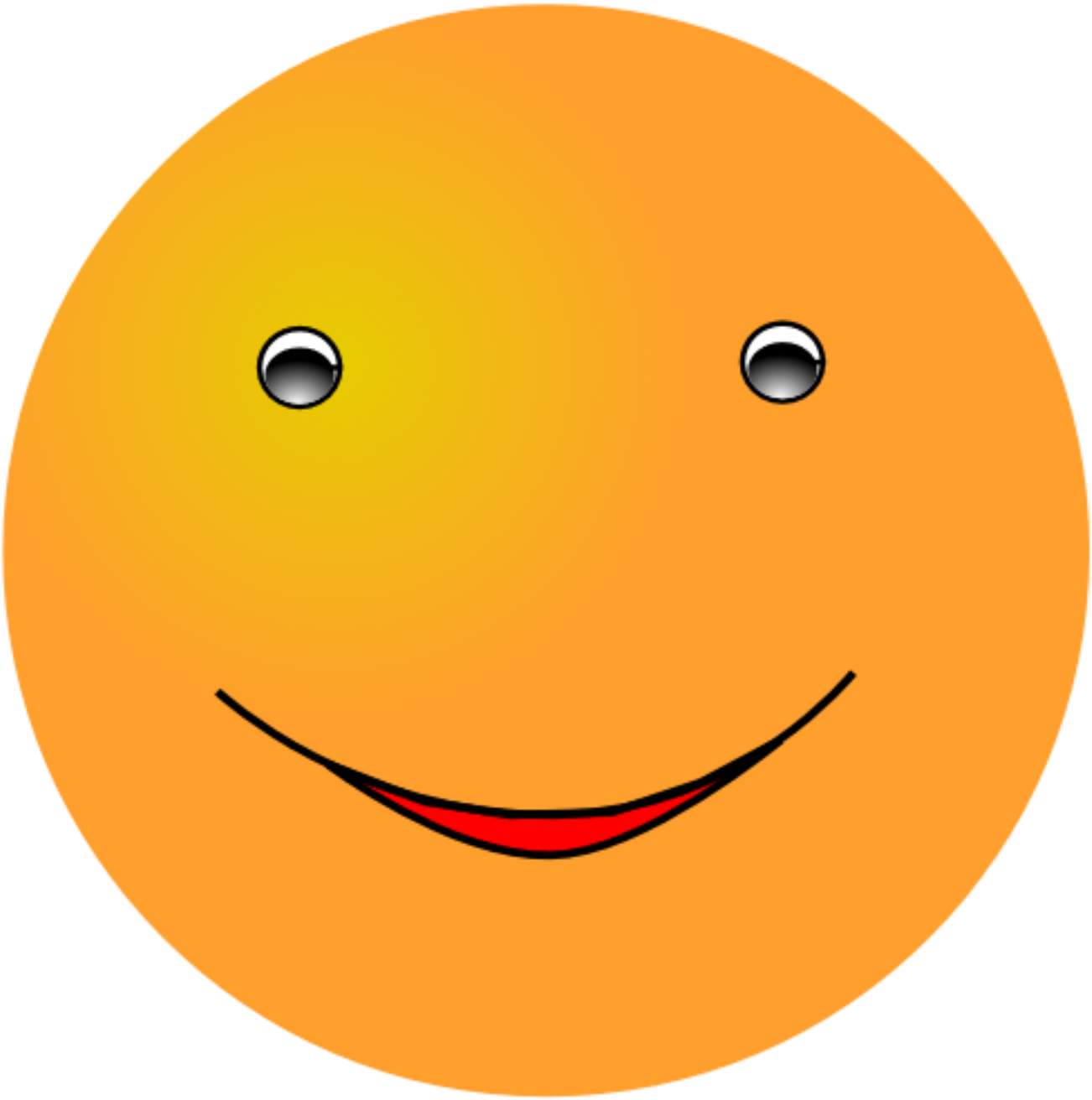
```
Starting tests...
Wed Nov 13 21:27:01 IST 2013
e1ef6e320d0e71fa999fb867ba8009a92ee33b8f  -


README
battleship.py
ex4.py

Testing README...
Done testing README...

Testing ex4.py...
result_code    bubble    20    1
result_code    choose    20    1
result_code    badinputs    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    badboards    6    1
result_code    boards    13    1
result_code    fireillegal    47    1
result_code    place    973    1
result_code    firemiss    113    1
result_code    firehit    16    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

```
roigreenberg
305571234
roi greenberg

================================
=  README for ex4: Pensioners  =
================================



==================
=  Description:  =
==================

ex4: Task 1: The function calculate (in arithmetics way) how much expences can be wasted any year
             in given amount of year with constant rate so at the end of your life
             you left with no money.
     Task 2: The function execute a bubble sort for sorting a list from minimum to maximum
             according to the second value.
     Task 3: The function find the most expensive house you can afford so you won't be in debts.
     Task 5: The functions The function find the most fun house you can afford so you won't
             be in debts.
battleship: some function for Battleship game.
     Task 1: The function create a new board in a given size. If no height given, create a square,
             If nothing given create 10X10 board
     Task 2: The function try to place a ship on the board if it not step-out from the board and not
             and not placed in another ship place.
     Task 3: The function fire on the board and try to destroy the ships

=============================
=  List of submitted files: =
=============================

README        This file
ex4.py            retirement calculation (expenses and houses) and sorting function.
battleship.py   execute Battleship game. create a board, place ships and
                fire on the ships and try do desrtoy them.

======================
=  Special Comments  =
======================
In ex4 task 1 I choose to use the arithmetics way
```

# 5 battleship.py

```
1  ##################################################################
2  # FILE: battleship.py
3  # WRITER: Roi Greenberg + roigreenberg + 305571234
4  # EXERCISE : intro2cs ex4 2013-2014
5  # Description: execute Battleship game. create a board, place
6  # ships and fire on the ships and try do desrtoy them
7  ##################################################################
8
9  def ship_index(board):
10     """find the highest ship index on board
11
12     Args:
13     -board: battleshipe board - you can assume its legal
14
15     return: the next ship-index."""
16
17     index = 0
18     # scaning the board for ships and remember the highest value
19     for wid in range(len(board[0])):
20         for heig in range(len(board)):
21             if board[heig][wid] is not None: # if ship found, take the higher index
22                 index = max(index, board[heig][wid][0])
23     return (index + 1)
24
25
26
27 def new_board(width=10,height="height"):
28     """creates a new board game for a Battleship game.
29
30     Args:
31     -width: a positive int - the width of the board - default value 10
32     -height: a positive int - the height of the board - if not spcified
33     should be as width
34
35     return: a NEW enpty board - each inner arrays is a list of 'None's.
36
37     n case of bad input: values are out of range returns None
38
39     You can assume that the types of the input arguments are correct."""
40
41     if height=="height": # if no height given, give the width value
42         height = width
43
44     # verifies the input
45     if height <=0 or width <= 0:
46         return
47     # create the board
48     board =[]
49     for heig in range(height):  # rows
50         board.append([])
51         for wid in range(width):  # lines
52             board[heig].append(None)
53     return board
54
55 def place_ship(board,ship_length,bow,ship_direction):
56     """Put a new ship on the board
57
58     put a new ship (with unique index) on the board.
59     in case of successful placing edit the board according to the definitions
```

```
    in the ex description.

    Args:
    -board - battleshipe board - you can assume its legal
    -ship_length: a positive int the length of the ship
    -bow: a tuple of ints the index of the ship's bow
    -ship_direction: a tuple of ints representing the direction the ship
    is facing (dx,dy) - should be out of the 4 options(E,N,W,S):
    (1,0) -facing east, rest of ship is to west of bow,
    (0,-1) - facing north, rest of ship is to south of bow, and etc.

    return: the index of the placed ship, if the placement was successful,
    and 'None' otherwise.

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

     You can assume the board is legal. You can assume the other inputs
     are of the right form. You need to check that they are legal."""

    # verifies the input
    if abs(ship_direction[0])+abs(ship_direction[1])==1 and \
       0 <= bow[0] < len(board[0]) and 0 <= bow[1] < len(board) and \
       -1 <= (bow[0] - ship_direction[0]*ship_length) <= len(board[0]) and \
       -1 <= (bow[1] - ship_direction[1]*ship_length) <= len(board):

        index=ship_index(board)  # find the next ship-index
        size=[ship_length]
        for part in range(ship_length):  # try to place the ship
            if board[bow[1]-ship_direction[1]*part]\
               [bow[0]-ship_direction[0]*part] == None:
                board[bow[1]-ship_direction[1]*part]\
                     [bow[0]-ship_direction[0]*part]  = (index, part, size)
            else:  # if another ship in the middle, delete the part of the ship
                   # alredy placed and return None
                for del_part in range(part):
                    board[bow[1]-ship_direction[1]*del_part]\
                         [bow[0]-ship_direction[0]*del_part] = None
                return
        return index


def fire(board,target):
    """implement a fire in battleship game

    Calling this function will try to destroy a part in one of the ships on the
    board. In case of successful fire destroy the relevant part
    in the damaged ship by deleting it from the board. deal also with the case
    of a ship which was completely destroyed

    -board - battleshipe board - you can assume its legal
    -target: a tuple of ints (x,y) indices on the board
    in case of illegal target return None

    returns: a tuple (hit,ship), where hit is True/False depending if the the
    shot hit, and ship is the index of the ship which was completely
    destroyed, or 0 if no ship was completely destroyed. or 0 if no ship
    was completely destroyed.

    Return None in case of bad input

    You can assume the board is legal. You can assume the other inputs
    are of the right form. You need to check that they are legal."""

    # verifies the input
    if target[0] < 0 or target[0] > len(board[0]) - 1 or \
       target[1] < 0 or target[1] > len(board) - 1:
        return
```

```
128
129     if board[target[1]][target[0]] is None: # when miss
130         return False, 0
131     else: # when hit
132         destroyed_index = 0 # 0 if the ship didn't destroy completely
133         # reduce the size of the ship
134         board[target[1]][target[0]][2][0] = board[target[1]][target[0]][2][0] \
135                                             - 1
136         # check if the ship have been destroyed completely
137         if board[target[1]][target[0]][2][0] == 0:
138             # recieve the index of the destroyed ship
139             destroyed_index = board[target[1]][target[0]][0]
140         board[target[1]][target[0]] = None  # remove the ship part from board
141     return True, destroyed_index
142
143
144
```

-1. You use magic numbers. It would be more clear to assign variables with meaningful names to hold the values of the indices you use. Without meaningful names, it isn't clear what information you're accessing. (For instance, there's no way to know what's in board[target[1]][target[2]][2][0]).

# 6 ex4.py

```
1  ##############################################################
2  # FILE: ex3.py
3  # WRITER: Roi Greenberg + roigreenberg + 305571234
4  # EXERCISE : intro2cs ex4 2013-2014
5  # Description:  retirement calculation (expenses and houses)
6  # and sorting function
7  ##############################################################
8
9  # calculate pension with variable growth rates
10 def variable_pension(salary, save, growth_rates):
11     """ calculate retirement fund assuming constant pesnion
12
13     A function that calculates the value of a retirement fund in each year
14     based on the worker salary, savings, working years and assuming constant
15     growthRate of the fund
16
17     Args:
18     - salary: the amount of money you earn each year,
19           a non negative float.
20     - save: the percent of your salary to save in the investment account
21            each working year -  a non negative float between 0 and 100
22     - growth_rate: the annual percent increase/decrease in your investment
23           account, a float larger than or equal to -100 (minus 100)
24     - years: number of years to work - non negative int
25
26     return: a list whose values are the size of your retirement account at
27       the end of each year.
28
29     In case of bad input: values are out of range
30     returns None
31
32     You can assume that the types of the input arguments are correct. """
33     GROWTH = 0.01
34     # verifies the input
35     for rate in growth_rates:
36         if float(rate) < -100:
37             return
38     if salary < 0 or save < 0 or save > 100:
39         return
40
41     if not growth_rates: # return empty list if no growth rates given
42         return []
43
44     # calculate the pention
45     pension = [salary * save * GROWTH]
46
47     # run for the length of the growth rates list
48     for i in range(1, len(growth_rates)):
49         pension.append(pension[i - 1]*(1 + float(growth_rates[i])*\
50                        GROWTH) + salary*save*GROWTH)
51
52     return pension  # return list of the pension value for each year
53
54
55
56 # calculate the retirement savings
57 def post_retirement(savings, growth_rates, expenses):
58     """ calculates the account status after retirement
59
```

```
    A function that calculates the account status after retirement, assuming
    constant expenses and no income
    Args:
    -savings: the initial amount of money in your savings account.
    A float larger than 0
    - growth_rates: a list of annual growth percentages in your investment
    account - a list of floats larger than or equal to -100.
    -expenses: the amount of money you plan to spend each year during
    retirement. A non negative float

    return: a list of your retirement account value at the end of each year.

    Note in case of a negative balance - the growth rate will change into
    rate on the debt
    In case of bad input: values are out of range returns None

    You can assume that the types of the input arguments are correct."""

    GROWTH = 0.01

    # verifies the input
    if not growth_rates:
        return []
    for rate in growth_rates:
        if rate < -100:
            return
    if savings < 0 or expenses < 0:
        return

    # Calculate the retirement saving
    # and put the into the new list
    retirement = [savings*(1+float(growth_rates[0])*GROWTH)-expenses]
    for i in range(1, len(growth_rates)):
        retirement.append(retirement[i-1]*(1+float(growth_rates[i]))\
                                          *GROWTH)-expenses)

    return retirement  # return list of the retirement saving in each year




# Find the maximal expenses you may expend during your lifetime
def live_like_a_king(salary, save, pre_retire_growth_rates,
                     post_retire_growth_rates, epsilon):
    """ Find the maximal expenses you may expend during your lifetime

    A function that calculates what is the maximal annual expenses you may
    expend each year and not enter into debts
    You may Calculate it using binary search or using arithmetics
    Specify in your README in which method you've implemnted the function

    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
```

```
128
129         You can assume that the types of the input arguments are correct."""
130         GROWTH = 0.01
131         # verifies the input
132         for rate in pre_retire_growth_rates:
133             if rate < -100:
134                 return
135         for rate in post_retire_growth_rates:
136             if rate < -100:
137                 return
138         if salary < 0 or save < 0 or save > 100 or epsilon <= 0:
139             return
140         # return None if no growth rates given
141         if len(post_retire_growth_rates) == 0:
142             return
143         # return "0.0" if no growth rates given
144         if len(pre_retire_growth_rates) == 0:
145             return 0.0
146
147         # calculate your savings in your retirement day
148         savings = variable_pension(salary, save, pre_retire_growth_rates)[-1]
149
150         # calculate the maximun expenses so you wont get in debts
151         sum1, sum2 = savings*(1+float(post_retire_growth_rates[0])*GROWTH), 1
152         for rate in post_retire_growth_rates[1:]:
153             sum1=sum1*(1+rate*GROWTH)
154             sum2=sum2*(1+rate*GROWTH)+1
155         expenses = float(sum1)/float(sum2)
156         return expenses
157
158
159
160
161
162
163
164
165     def bubble_sort_2nd_value(tuple_list):
166         """sort a list of tuples using bubble sort algorithm
167
168         Args:
169         tuples_list - a list of tuples, where each tuple is composed of a string
170         value and a float value - ('house_1',103.4)
171
172         Return: a NEW list that is sorted by the 2nd value of the tuple,
173         the numerical one. The sorting direction should be from the lowest to the
174         largest. sort should be stable (if values are equal, use original order)
175
176         You can assume that the input is correct."""
177
178         reordered_list=tuple_list[:] # copy the value to new list
179         for i in range(len(reordered_list)):  # bubble sort according to 2nd value
180             for  j in range (0,len(reordered_list)-i-1):
181                 if reordered_list[j][1] > reordered_list[j+1][1]:
182                     reordered_list[j+1], reordered_list[j] =  \
183                     reordered_list[j], reordered_list[j+1]
184
185         return reordered_list
186
187
188
189
190
191     def choosing_retirement_home(savings,growth_rates,retirement_houses):
192         """Find the most expensive retirement house one can afford.
193
194         Find the most expensive, but affordable, retiremnt house.
195         Implemnt the function using binary search
```

When your function is meant to return None in case of an error, it's better to write explicitly 'return none'. It's more clear that way.

+5 bonus.

```
196
197         Args:
198         -savings: the initial amount of money in your savings account.
199         -growth_rates: a list of annual growth percentages in your
200         investment account - a list of floats larger than or equal to -100.
201         -retirement_houses: a list of tuples of retirement_houses, where
202         the first value is a string - the name of the house and the
203         second is the annual rent of it - nonnegative float.
204
205         Return: a string - the name of the chosen retirement house
206         Return None if can't afford any house.
207
208         You need to test the legality of savings and growth_rates
209         but you can assume legal retirement_house list
210         You can assume that the types of the input are correct"""
211
212
213         # verifies the input
214         if not growth_rates:
215             return
216         for rate in growth_rates:
217             if rate < -100:
218                 return
219         if savings < 0 or not retirement_houses:
220             return
221
222         # sort the houses according to their annual rental
223         reorder_houses = bubble_sort_2nd_value(retirement_houses)
224
225         # return None if no house affordable
226         if post_retirement(savings, growth_rates, reorder_houses[0][1])[-1] < 0:
227             return
228         # check if all the houses affordable. if so, choose the last house
229         elif post_retirement(savings, growth_rates, reorder_houses[-1][1])[-1] > 0:
230             best_house = len(reorder_houses)-1
231         else:
232             # search for the best affordable house using binary-search
233             cheap_house, exp_house = 0, len(reorder_houses)
234             best_house = (cheap_house+exp_house)//2
235             while best_house != cheap_house:
236                 saving_last_year = post_retirement(savings, growth_rates, \
237                                         reorder_houses[best_house][1])[-1]
238                 best_house_cost = reorder_houses[best_house]
239                 if saving_last_year >= 0:
240                     cheap_house = best_house
241                 elif saving_last_year < 0:
242                     exp_house = best_house
243                 best_house = (cheap_house+exp_house)//2
244
245         # check for former house with same annual rental
246         while reorder_houses[best_house][1] == reorder_houses[best_house-1][1] \
247               and best_house!=0:
248             best_house -= 1
249
250         return reorder_houses[best_house][0]
251
252
253
254     def get_value_key(value=0):
255         """returns a function that calculates the new value of a house
256
257
258         #Args:
259         -value: the value added per opponent - a float - the default value is 0
260
261         This function returns a function that accepts triple containing
262         (house ,anntual rent,number of opponents) and returns the new value of
263         this house - annual_rent+value*opponents
```

You could have used the 'live_like_a_king' function to efficiently find the amount of money you have to spend.

```
264
265         You can assume that the input is correct."""
266
267         def new_rent(house):
268             """return the fun value of the house
269
270             #Args:
271             -house:(tuple), where  the first value
272             is a string - the name of the house,
273             the second is the annual rent on it - a non negative float, and the
274             third is the number of battleship opponents the home hosts - non
275             negative int"""
276
277             return house[1]+value*house[2]
278
279         return new_rent
280
281
282
283
284
285 def choose_retirement_home_opponents(budget,key,retirement_houses):
286     """ Find the best retiremnt house that is affordable and fun
287
288     A function that returns the best retiremnt house to live in such that:
289     the house is affordable and
290     his value (annual_rent+value*opponents) is the highest
291
292     Args:
293     -annual_budget: positive float. The amount of money you can
294     expand per year.
295     -key: a function of the type returned by get_value_key
296     -retirement_houses: a list of houses (tuples), where  the first value
297     is a string - the name of the house,
298     the second is the annual rent on it - a non negative float, and the third
299     is the number of battleship opponents the home hosts - non negative int
300
301     Returns the name of the retirement home which provides the best value and
302     which is affordable.
303
304     You need to test the legality of annual_budget,
305     but you can assume legal retirement_house list
306     You can assume that the types of the input are correct"""
307
308     # verifies the input
309     if budget <= 0:
310         return
311     if not retirement_houses:
312         return
313
314     # sort the houses according to their fun value
315     sorted_retirement_houses=sorted(retirement_houses, key=key)
316     # search and return the most fun house that affordable
317     # return None if none affordable
318     for best_house in range(len(sorted_retirement_houses)):
319         if sorted_retirement_houses[-1 - best_house][1] < budget:
320             return sorted_retirement_houses[-1-best_house][0]
```

-1. More magic numbers (also in the previous function).