**Name**: Steve Hommy

**Pair: -**

**Amount of completed tasks: 10**

**Which tasks were left undone or incomplete: 0**

Self-assessment:

This exercise was kind of hard for me because working with dictionary is not that easy. Doing this exercise, I learned how dictionary really work. I am still wondering if I could work with dictionary more efficiently. I understood the questions and managed to do my tasks

## Test report

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | **Input / action** | **Desired output** | **Actual output (use red color if desired output != actual output)** |
| **2** | User runs the program  <Run the program a couple of times so that you get **each side up** at least once> | This side is up: Heads  Tossing the coin…  Now this side is up: <Tails>  <Heads, Tails, Rabbit whole, Upright, wormhole> | This side is up: Heads  Tossing the coin...  Now this side is up: Heads  Currency is: Dollar |
| **2** | User runs the program  <Run the program a couple of times so that you get **each currency** at least once.> | Currency is: Euro  <Euro, Pound, Dollar, Ruble, Yen> | This side is up: Heads  Tossing the coin...  Now this side is up: Tails  Currency is: Ruble |
|  |  |  |  |
| **3** | User runs the program  <Run the program a couple of times so that you get **each currency** at least once.> | Currency (original): Euro  Currency (new): <Dollar>  <Euro, Pound, Dollar, Ruble, Yen> | This side is up: Heads  Tossing the coin...  Now this side is up: Coin defies gravity and gets lost on a wormhole in space  Original currency Euro  New currency: Pound |
|  |  |  |  |
| **5** | User runs the program  <Run the program a couple of times so that you get **each number, color and that extra feature** at least once> | Rolling the dice …  number: 4  color: red  extra feature: xx  <1..6, all the colors, all the feature values> | Rolling the dice…  Number: 6  Color: Blue  Icon: ⚅ |
|  |  |  |  |
| **6** | User runs the program  <Run the program a couple of times so that you get **each possible sum** at least once> | Rolling the dice1 …  number: 4  Rolling the dice2 …  number: 2  The sum is: 6  The sum is <2..12> | Rolling the dice1…  Number: 5  Rolling the dice2…  Number: 6  The sum is: 11 |
|  |  |  |  |
| **7** | User runs the program  <Run the program a couple of times so that you get **every player to win** at least once.> | Dice rolling game  First round …  Player1: 6  Player2: 4  Player3: 5  Second round …  Player1: 3  Player3: 5  The winner is: Player3  <Player1, Player2, Player3> | {'Player1': 2, 'Player2': 4, 'Player3': 6}  {'Player2': 4, 'Player3': 6}  The winner is: Player3  {'Player1': 6, 'Player2': 6, 'Player3': 5}  Player1 and player2 have to roll again  Player1: 6  Player2: 1  {'Player1': 6, 'Player3': 5}  The winner is: Player1 |
|  |  |  |  |
| **8** | User runs the program  <Run the program a couple of times so that you get **every player to win** at least once.> | Dice rolling game  First round …  Player1: 6  Player2: 6  Player3: 5  Player 2 is out because of red dice.  Second round …  Player1: 3  Player3: 6  The winner is: Player3  <Player1, Player2, Player3> | {'Player1': 2, 'Player2': 2, 'Player3': 1}  Player1: 1  Player2: 3  Player2 unfortunately got red dice and loses  {'Player1': 2, 'Player3': 1}  Player1 and player3 have to roll again  Player1: 6  Player3: 1  The winner is: Player1 |
|  |  |  |  |
| **9** | User runs the program  <Write test case depending on your implementation.> | Here is the data that you provided :  Manufacturer: <Apple>  Model number: <iPhone 7>  Retail price: <500.0> | Enter manufacturer: Apple  Enter model: Iphone 7  Enter retail price: 500  Here is the data that you provided:  Manufacturer: Apple  Model number: Iphone 7  Retail price: 500 |

1. Explain the following terms:

a. Abstraction (in programming)

**Abstraction is used to hide background details or any unnecessary implementation about the data so that users only see the required information. So, in simple terms, abstraction “displays” only the relevant attributes of objects and “hides” the unnecessary details.**

b. Accessor and mutator methods

**Accessor method is used to access the state of the object. With it the data hidden in the object can be accessed from this method. However, this method cannot change the state of the object, it can only access the data hidden. We can name these methods with the word (get).**

**Mutator method is used to mutate/modify the state of an object. It alters the hidden value of the data variable. It can set the value of a variable instantly to a new value. This method is also called as update method. Moreover, we can name these methods with the word (set).**

c. Public and private methods

**Public method makes property/method available from anywhere. It is accessible both inside and outside the scope of class. Any instance of that class will have access to public methods and can invoke them.**

**Private methods when you want your property/method to be visible in its own class only. It’s not accessible outside the scope of the class whatsoever. If you try to invoke a private method with a class object, an error will occur.**

d. \_\_str\_\_ method (in Python)

**The \_\_str\_\_ method should be defined in a way that is easy to read and outputs all the members of the class. This method is also used as a debugging tool when the members of a class need to be checked. The \_\_str\_\_ method is called when the following functions are invoked on the object and return a string: print(), str()**

5. Create a class Dice and make an object of it. You shall be able to roll the dice, get the result (number between 1 – 6) and get its color. Add at least 1 extra feature. Design your program using pseudocode. Document your code properly (with good comments) and pay attention to the clarity of the output prints.

Import random module

Class Dice:

Define initialize self

Number

Color

Icon

Define dice rolling self

Random number variable that is random integer between 1 to 6

If random number is 1

Self number is 1

Self color is black

Self icon is ⚀

Else if random number is 2

Self number is 2

Self color is white

Self icon is ⚁

Else if random number is 3

Self number is 3

Self color is green

Self icon is ⚂

Else if random number is 4

Self number is 4

Self color is yellow

Self icon is ⚃

Else if random number is 5

Self number is 5

Self color is red

Self icon is ⚄

Else

Self number is 6

Self color is blue

Self icon is ⚅

Define getting number self

Return self number

Define getting color self

Return self color

Define getting icon self

Return self icon

Define main

Variable that has value of class Dice

Run dice rolling function

Print rolling the dice

Print get number function

Print get color function

Print get icon function

Run main function

7. Design first using pseudocode, then code this: Create a Dice rolling game of three players (three Dice objects). On first round everybody rolls their dice, lowest number loses and is out of game. On second round the two remaining contestants roll a dice and higher number wins. Use proper output prints of the situation all the time. If on either round there is a tie between 2 or 3 dices, then the tied dices are rolled again.

Import random

Class Dice

Define initialize(self)

Self.number1 = 1

Self.number2 = 1

Self.number3 = 1

Define player1\_roll(self) **(Do same for player2 and player3)**

Player1\_dice = random.randint(1, 6)

Self.number1 = player1\_dice

Define get\_player1(self) **(Do same for player2 and player3)**

Return integer(self.number1)

Create 3 dice objects

Call all 3 player\_roll() with their specific object

Create player dictionary where every player have their own get\_player value

Define check\_duplicates\_and\_remove\_player

While length of dictionary is larger than 1

Check with If and elif if any of the player have the same value if they do then they have to roll again and the one with smaller value will be removed using pop() method

Else

player\_dict.pop(min(player\_dict, key=player\_dict.get))

define main

call check\_duplicates\_and\_remove\_player

for player in player\_dict.keys():

call main

9. Design first using pseudocode, then code this: Create a CellPhone Class. Write a program for a class that represents a cell phone. The data attributes are manufact (Manufacter), model (Model) and retail\_price (Retail price). The class will also have the following methods: • \_\_init\_\_ • set manufact • set model • set retail price • get manufact • get model • get retail price

Class Cellphone

Define initialize(self)

Manufact = “”

Model = “”

Retail price = 0

Define set manufact

Manufact = string(user input())

Define set model

Model = string(user input())

Define set retail price

Retail price = integer(user input())

Define get manufact

Return manufact

Define get model

Return model

Define get retail price

Return retail price

Define main

Cellphone = Cellphone()

Call all set functions using cellphone variable

Print Here is the data that you provided

Print all get functions using cellphone variable

10. Take a look at the CellPhone Class/Object: where are these concepts (or are they there) (take a screen capture and indicate a line)?

a. Object?

b. Encapsulation?

c. Data attributes?

d. Hidden attributes?

e. Public methods?

f. Private methods?

g. Init-method?



# Screen captures of all code here

# File name: exercise2-4.py

# Author: Steve Hommy

# Description: Coin tosser

import random

# Class definition and initializing attributes. sideup attribute is private

class Coin:

    def \_\_init\_\_(self):

        self.\_\_sideup = "Heads"

        self.currency = "Euro"

    # Defining toss\_the\_coin

    # Getting random integer between 0 to 4

    # If value is 0 sideup is heads, elif 1 sideup is tails, elif 2 sideup is Coin lands on the table upright,

    # elif 3 sideup is coin drops on the ground and disappears on a rabbit hole

    # else Coin defies gravity and gets lost on a wormhole in space

    def toss\_the\_coin(self):

        random\_number = random.randint(0,4)

        if random\_number == 0:

            self.\_\_sideup = "Heads"

        elif random\_number == 1:

            self.\_\_sideup = "Tails"

        elif random\_number == 2:

            self.\_\_sideup = "Coin lands on the table upright"

        elif random\_number == 3:

            self.\_\_sideup = "coin drops on the ground and disappears on a rabbit hole"

        else:

            self.\_\_sideup = "Coin defies gravity and gets lost on a wormhole in space"

    # Defining get\_sideup and returning sideup value

    def get\_sideup(self):

        return self.\_\_sideup

    # Defining generate\_the\_currency.

    # List of different currency and return randomly selected element using choice() method

    def generate\_the\_currency(self):

        random\_currency = random.choice(["Euro", "Pound", "Dollar", "Ruble", "Yen"])

        self.currency = random\_currency

    # Defining get\_currency and returning currency value

    def get\_currency(self):

        return self.currency

# Main function definition

def main():

    # Creating object and calling the functions

    my\_coin = Coin()

    print("This side is up:", my\_coin.get\_sideup())

    print("Original currency", my\_coin.get\_currency())

    print("Tossing the coin...")

    my\_coin.toss\_the\_coin()

    print("Now this side is up:", my\_coin.get\_sideup())

    my\_coin.generate\_the\_currency()

    print("Currency is:", my\_coin.get\_currency())

# Calling the main function

main()

# File name: exercise5.py

# Author: Steve Hommy

# Description: Dice roller

import random

# Class definition and initializing attributes.

class Dice:

    def \_\_init\_\_(self):

        self.number = 1

        self.color = "Black"

        self.icon = "⚀"

    # Defining roll\_the\_dice. Getting random integer between 1 to 6.

    # Using if, elif and else statment to choose specific value

    def roll\_the\_dice(self):

        random\_number = random.randint(1, 6)

        if random\_number == 1:

            self.number = 1

            self.color = "Black"

            self.icon = "⚀"

        elif random\_number == 2:

            self.number = 2

            self.color = "White"

            self.icon = "⚁"

        elif random\_number == 3:

            self.number = 3

            self.color = "Green"

            self.icon = "⚂"

        elif random\_number == 4:

            self.number = 4

            self.color = "Yellow"

            self.icon = "⚃"

        elif random\_number == 5:

            self.number = 5

            self.color = "Red"

            self.icon = "⚄"

        else:

            self.number = 6

            self.color = "Blue"

            self.icon = "⚅"

    # Defining functions and returning their values

    def get\_number(self):

        return self.number

    def get\_color(self):

        return self.color

    def get\_icon(self):

        return self.icon

# Defining main

def main():

    # Creating object and calling the functions

    dice = Dice()

    dice.roll\_the\_dice()

    print("Rolling the dice…")

    print("Number:", dice.get\_number())

    print("Color:", dice.get\_color())

    print("Icon:", dice.get\_icon())

main()

# File name: exercise6.py

# Author: Steve Hommy

# Description: Sum of 2 dice value

import random

# Class definition and initializing attributes.

class Dice:

    def \_\_init\_\_(self):

        self.number = 1

    # Defining roll\_the\_dice for 2 dice where they get random integer between 1 to 6

    # and returning their values

    def roll\_the\_dice1(self):

        random\_number = random.randint(1, 6)

        self.number = random\_number

    def get\_dice1(self):

        return self.number

    def roll\_the\_dice2(self):

        random\_number = random.randint(1, 6)

        self.number = random\_number

    def get\_dice2(self):

        return self.number

def main():

    # Creating 2 objects, calling the functions and sum of two dice

    dice1 = Dice()

    dice2 = Dice()

    dice1.roll\_the\_dice1()

    dice2.roll\_the\_dice2()

    print("Rolling the dice1…\nNumber:", dice1.get\_dice1())

    print("Rolling the dice2…\nNumber:", dice2.get\_dice2())

    print("The sum is:", int(dice1.get\_dice1()) + int(dice2.get\_dice2()))

main()

# File name: exercise7.py

# Author: Steve Hommy

# Description: Dice rolling game of three players

import random

# Creating class and initializing attributes

class Dice:

    def \_\_init\_\_(self):

        self.number1 = 1

        self.number2 = 1

        self.number3 = 1

    # Get random integer between 1 to 6 and setting the value for each player

    def player1\_roll(self):

        player1\_dice = random.randint(1, 6)

        self.number1 = player1\_dice

    def get\_player1(self):

        return int(self.number1)

    def player2\_roll(self):

        player2\_dice = random.randint(1, 6)

        self.number2 = player2\_dice

    def get\_player2(self):

        return int(self.number2)

    def player3\_roll(self):

        player3\_dice = random.randint(1, 6)

        self.number3 = player3\_dice

    def get\_player3(self):

        return int(self.number3)

# Creating three object

dice1 = Dice()

dice2 = Dice()

dice3 = Dice()

# Calling the roll function with specific object

dice1.player1\_roll()

dice2.player2\_roll()

dice3.player3\_roll()

# Creating dictionary and adding values for players

player\_dict = {"Player1": dice1.get\_player1(), "Player2": dice2.get\_player2(), "Player3": dice3.get\_player3()}

def check\_duplicates\_and\_remove\_player():

    # While dictionary length is larger than 1 then keep running.

    # Checking equal values between players, when value is equal then roll again.

    # Player with lowest value will be removed from dictionary with pop() method

    while len(player\_dict) > 1:

        print(player\_dict)

        if dice1.get\_player1() == dice2.get\_player2():

            print("Player1 and player2 have to roll again")

            dice1.player1\_roll()

            dice2.player2\_roll()

            print("Player1:", dice1.get\_player1())

            print("Player2:", dice2.get\_player2())

            if dice1.get\_player1() > dice2.get\_player2():

                player\_dict.pop("Player2")

            else:

                player\_dict.pop("Player1")

        elif dice1.get\_player1() == dice3.get\_player3():

            print("Player1 and player3 have to roll again")

            dice1.player1\_roll()

            dice2.player3\_roll()

            print("Player1:", dice1.get\_player1())

            print("Player3:", dice1.get\_player3())

            if dice1.get\_player1() > dice2.get\_player3():

                player\_dict.pop("Player3")

            else:

                player\_dict.pop("Player1")

        elif dice2.get\_player2() == dice3.get\_player3():

            print("Player2 and player3 have to roll again")

            dice1.player2\_roll()

            dice2.player3\_roll()

            print("Player2:", dice2.get\_player2())

            print("Player3:", dice1.get\_player3())

            if dice1.get\_player2() > dice2.get\_player3():

                player\_dict.pop("Player3")

            else:

                player\_dict.pop("Player2")

        elif dice1.get\_player1() == dice2.get\_player2() == dice3.get\_player3():

            print("All players have to roll again")

            dice1.player1\_roll()

            dice1.player2\_roll()

            dice2.player3\_roll()

        else:

            # The min function returns the minimum value of an iterable according to the given key.

            # In this case it returns the key of player\_dict with the minimum value.

            # player\_dict.get allows you to access the corresponding value to the dictionary key,

            # which are iterated over when you iterate over player\_dict.

            # The key argument to the min specifies what key you want to find the minimum on.

            player\_dict.pop(min(player\_dict, key=player\_dict.get))

# Defining main

def main():

    # Calling check\_duplicates\_and\_remove\_player

    check\_duplicates\_and\_remove\_player()

    # Iterating over dictionary keys()

    for player in player\_dict.keys():

        print("The winner is:", player)

# Calling main

main()

# File name: exercise8.py

# Author: Steve Hommy

# Description: Dice rolling game of three players

import random

# Creating class and initializing attributes

class Dice:

    def \_\_init\_\_(self):

        self.number1 = 1

        self.number2 = 1

        self.number3 = 1

    # Get random integer between 1 to 6 and setting the value for each player

    def player1\_roll(self):

        player1\_dice = random.randint(1, 6)

        self.number1 = player1\_dice

    def get\_player1(self):

        return int(self.number1)

    def player2\_roll(self):

        player2\_dice = random.randint(1, 6)

        self.number2 = player2\_dice

    def get\_player2(self):

        return int(self.number2)

    def player3\_roll(self):

        player3\_dice = random.randint(1, 6)

        self.number3 = player3\_dice

    def get\_player3(self):

        return int(self.number3)

# Creating three object

dice1 = Dice()

dice2 = Dice()

dice3 = Dice()

# Calling the roll function with specific object

dice1.player1\_roll()

dice2.player2\_roll()

dice3.player3\_roll()

# Creating dictionary and adding values for players

player\_dict = {"Player1": dice1.get\_player1(), "Player2": dice2.get\_player2(), "Player3": dice3.get\_player3()}

def check\_duplicates\_and\_remove\_player():

    # While dictionary length is larger than 1 then keep running.

    # Checking equal values between players, when value is equal then winner will be decided by color of the dice.

    # Player with lowest value or with red dice will be removed from dictionary with pop() method

    while len(player\_dict) > 1:

        print(player\_dict)

        if dice1.get\_player1() == dice2.get\_player2():

            dice1.player1\_roll()

            dice2.player2\_roll()

            player1\_color = random.choice(["Red", "Blue"])

            if player1\_color == "Red":

                print("Player1 unfortunately got red dice and loses")

                player\_dict.pop("Player1")

            else:

                print("Player2 unfortunately got red dice and loses")

                player\_dict.pop("Player2")

        elif dice1.get\_player1() == dice3.get\_player3():

            dice1.player1\_roll()

            dice2.player3\_roll()

            player3\_color = random.choice(["Red", "Blue"])

            if player3\_color == "Red":

                print("Player3 unfortunately got red dice and loses")

                player\_dict.pop("Player3")

            else:

                print("Player1 unfortunately got red dice and loses")

                player\_dict.pop("Player1")

        elif dice2.get\_player2() == dice3.get\_player3():

            dice1.player2\_roll()

            dice2.player3\_roll()

            player2\_color = random.choice(["Red", "Blue"])

            if player2\_color == "Red":

                print("Player2 unfortunately got red dice and loses")

                player\_dict.pop("Player2")

            else:

                print("Player3 unfortunately got red dice and loses")

                player\_dict.pop("Player3")

        elif dice1.get\_player1() == dice2.get\_player2() == dice3.get\_player3():

            print("All players have to roll again")

            dice1.player1\_roll()

            dice1.player2\_roll()

            dice2.player3\_roll()

        else:

            # The min function returns the minimum value of an iterable according to the given key.

            # In this case it returns the key of player\_dict with the minimum value.

            # player\_dict.get allows you to access the corresponding value to the dictionary key,

            # which are iterated over when you iterate over player\_dict.

            # The key argument to the min specifies what key you want to find the minimum on.

            player\_dict.pop(min(player\_dict, key=player\_dict.get))

# Defining main

def main():

    # Calling check\_duplicates\_and\_remove\_player

    check\_duplicates\_and\_remove\_player()

    # Iterating over dictionary keys()

    for player in player\_dict.keys():

        print("The winner is:", player)

# Calling main

main()

# File name: exercise9.py

# Author: Steve Hommy

# Description: Create a CellPhone Class

class Cellphone:

    def \_\_init\_\_(self):

        self.manufact = ""

        self.model = ""

        self.retail\_price = 0

    def set\_manufact(self):

        self.manufact = str(input("Enter manufacturer: "))

    def set\_model(self):

        self.model = str(input("Enter model: "))

    def set\_retail\_price(self):

        self.retail\_price = int(input("Enter retail price: "))

    def get\_manufact(self):

        return self.manufact

    def get\_model(self):

        return self.model

    def get\_retail\_price(self):

        return self.retail\_price

def main():

    cellphone = Cellphone()

    cellphone.set\_manufact()

    cellphone.set\_model()

    cellphone.set\_retail\_price()

    print("Here is the data that you provided:")

    print("Manufacturer:", cellphone.get\_manufact())

    print("Model number:", cellphone.get\_model())

    print("Retail price:", cellphone.get\_retail\_price())

main()