COURSEWORK A DOCUMENTATION

Pseudocode and Code Explanation

21st November, 2019

Hue Nguyen
Student Id: 19035298
hue2.nguyen@live.uwe.ac.uk

Contents

I. Date.py, Restrict.py

A. Date.py

- The class Date checks if input date is correct and create a common format for date input (DD-MM-YYYY)
- function date_format(date) check if date input is of the right format (separated by a dash or a forward slash) and return an instance of Date.

```
class Date {
    attribute day of type int
    attribute month of type int
    attribute year of type int

    constructor (day with default value = 0, month with default value = 0,
                year with default value = 0) {
        if day is 0 and month is 0 and year is 0 then
            attribute day = 0
            attribute month = 0
            attribute year = 0

        else if month_check(month) is True and year_check(year) is True and
                day_check(month, day, year) then
            attribute day = day
            attribute month = month
            attribute year = year

        else
            display a message saying the date is of the wrong format or date
            does not exist
    }

    function month_check(month) {
        if (month is in range [1, 12]) and (month's number of digits is in
                range [1,2]) then
            return True
    }

    function year_check(year) {
        if year's number of digits is in range [1, 2] or equals to 4 then
            return True
    }

    function day_check(month, day, year) {
        if day's number of digits is larger than 2 then
            return False

        if month is 1 or 3 or 5 or 7 or 8 or 10 or 12 then
            if day is in range [1, 31] then
                return True

        if (year is leap) and (m is 2) then
            if day is in range [1,29] then
                return True
```

```
    }

    function __str__() {
        if day is 0 and month is 0 and year is 0 then
            return empty string
        else
            return a string displaying the date in format DD-MM-YYYY
    }
}

// function to format the date
function date_format(date) {
    if date is not empty then
        day, month, year = date splitted by a dash
        return an instance of Date(day, month, year)
    else
        return default Date()
}
```

B. Restrict.py

- the functions in this file will check if the user input is of the correct type.

```
function getter_setter_gen(attribute, type) {
    function getter() {
        return attribute
    }

    function setter(value)
        if value is not an instance of type then
            raise a TypeError
        else
            set the value to the attribute

    return a class property with getter and setter
}

function check_attribute(class) {
    attribute_dictionary = empty dictionary
    for every key-value pair in the buitin dictionary of the class do
        if value is an instance of type then
            value = call function getter_setter_gen(key, value)
        assign value to key in attribute_dictionary

     return a new class, replacing current class deictionary with the
     modified dictionary
}
```

II. Animal Class, Pet Class, Wild Animal Class, Treatment Class

A. Animal

```
class Animal {
    attribute sanctuary_id of type String
    attribute animal_type of type String
    attribute vaccinated of type String
    attribute reason_for_entry of type String
    attribute date_of_arrival of type Date
    attribute date_of_departure of type Date
    attribute destination of type String
    attribute destination_address of type String

    constructor (sanctuary_id, animal_type, vaccinated, reason_for_entry,
                 date_of_arrival, date_of_departure, destination,
                 destination_address)
        attribute sanctuary_id        = sanctuary_id
        attribute animal_type         = animal_type
        attribute vaccinated          = vaccinated
        attribute reason_for_entry    = reason_for_entry
        attribute date_of_arrival     = date_of_arrival
        attribute date_of_departure   = date_of_departure
        attribute destination         = destination
        attribute destination_address = destination_address

    function __str__() {
        return a string containing all information of an Animal instance

    }

}
```

B. Pet

```
class Pet inherits Animal {
    attribute breed of type String
    attribute neutered of type String
    attribute microchip of type String

    constructor (sanctuary_id, animal_type, vaccinated, reason_for_entry,
                 date_of_arrival, date_of_departure, destination,
                 destination_address, breed, neutered, microchip) {
        call constructor of Animal class
        attribute breed     = breed
        attribute neutered  = neutered
        attribute microchip = microchip
    }

    function __str__() {
        return a string containing all information of a Pet instance
    }
 }
```

C. Wild Animal

```
class Wild Animal inherits Animal{
    pass
}
```

D. Treatment

```
class Treatment {
    attribute sanctuary_id of type String
    attribute surgery of type String
    attribute surgery_date of type Date
    attribute medication of type String
    attribute medication_start of type Date
    attribute medication_finish of type Date
    attribute responsible_for_abuse of type String
    attribute responsible_for_abandoning of type String

    constructor (sanctuary_id, surgery, surgery_date, medication,
                 medication_start, medication_fin, responsible_for_abuse,
                 responsible_for_abandoning) {
        attribute sanctuary_id = sanctuary_id
        attribute surgery = surgery
        attribute surgery_date = surgery_date
        attribute medication = medication
        attribute medication_start = medication_start
        attribute medication_finish = medication_finish
        attribute responsible_for_abuse = responsible_for_abuse
        attribute responsible_for_abandoning = responsible_for_abandoning
    }

    function __str__() {
        return a string containing all information of a Treatment instance
    }
}
```

III. AVLTree

A. Main Methods

1. Insertion

The insertion process requires the following functions:

- `insert(object, object_type)`: this function calls the recursive function `insert_recur(root, node)` to find the correct position and balance the tree.
- `insert_recur(current_root, node)`: this function recursively goes through the tree to find the correct position for the `node`, then perform rotations to rebalance the tree.
- `right_rotate(node)`: performs right rotation around the `node`, see details below.
- `left_rotate(node)`: performs left rotation around the `node`, see details below.
- `get_height(node)`: returns the height of the `node` if it exists.
- `get_difference(node)`: returns the height difference between two branches of the `node`.

2. Search

The searching process requires the following functions:

- `search(key, object_type)`: this function calls the recursive function `search_recur(root, key)` to find the node with the passed in `key`.
- `search(current_root, key)`: this function recursively goes through the tree to find the node with the `key`.

3. **Deletion**

The deletion process requires the following functions:

- `delete(key, object_type)`: this function finds the animal using passed in `key` and, if found, calls the recursive function `delete_recur(root, aniaml_id)` to delete the node and rebalance the tree.
- `delete_recur(current_root, key)`: this function recursively searches for the node, perform standard BST deletion and rebalance the tree if necessary.
- `find_successor(node)`: this function is used to find the successor to the deleted `node`, returns the leftmost node.
- `right_rotate(node)`: performs right rotation around the `node`.
- `left_rotate(node)`: performs left rotation around the `node`.
- `get_height(node)`: returns the height of the `node` if it exists.
- `update_height(node)`: update the height of the `node`.
- `get_difference(node)`: returns the height difference between two branches of the `node`.

B. Implementation

```
class NodeAnimal{
    constructor(animal){
        attribute animal = animal
        attribute key     = san_id of animal
        attribute left   = None
        attribute right  = None
        attribute height = 1
    }
}

class NodePerson {
    constructor(person_name) {
        attribute key = person_name
        attribute left   = None
        attribute right  = None
        attribute height = 1
}

class AVLTree {
    constructor(){
        attribute root = None
    }

    // function to search for animal using animal id, this function calls
       the search_recur function to perform recursive search
    function search(key, object_type with empty string as default){
        node = call search_recur(self.root, key)

        if object_type is "person" then
            if node exists then
                return node's key
```

```
        else
            return a message saying person does not exit in database
    else
        if node is found then
            return animal of node
        else
            return a string saying animal is not found
}

function search_recur(current_root, key) {
    if current_root is None or key is at root then
        return current_root
    if key > the current_root's id then
        return search_recur(current_root's right node, key)
    if key < the current_root's id then
        return search_recur(current_root's left node, key)
}

// function to rotate left around a node
function left_rotate(node) {
    R = node's right child

    assign left branch of R to right of node
    assign node to left of R

    update height of node
    update height of R

    return R
}

// function to rotate right around a node
function right_rotate(node) {
    L = node's left child

    assign right brach of L to left of node
    assign node to right of L

    update height of node
    update height of L

    return L
}

// function to insert an object into tree, it calls the recursive
   function insert_recur to find the correct position and rebalance the
   tree
function insert(object, object_type with empty string as default) {
    if object_type is "person" then
        root = call insert_recur(root, NodePerson(object))
    else
        root = call insert_recur(root, NodeAnimal(animal))
}
```

```
function insert_recur(current_root, node) {
    if current_root does not exist then
        return node
    else if node's key > current_root's key then
        current_root's right = insert_recur(current_root's right, node)
    else if node's key < current_root's key then
        current_root's left = insert_recur(current_root's left, node)

    update height of current_root
    difference = get height difference between current's root children

    // Perform rotation to rebalance the tree
    if (current_root's left child exists)
            and (node's key < current_root's left child's key)
            and (height difference > 1) then
        return rotate right around current_root

    if (current_root's right child exists)
            and (node's key > current_root's right child's key)
            and (height difference < -1) then
        return rotate left around current_root

    if (current_root's left child exists)
            and (node's key > current_root's left child's key)
            and (height difference > 1) then
        current_root's left = rotate left around current_root's left child
        return rotate right around current_root

    if (current_root's right child exists)
            and (node's key < current_root's right child's key)
            and (height difference < -1) then
       current_root's right = rotate right around current_root's right child
       return rotate right around current_root

    return current_root
}


// function to delete a node, this function calls the recursive function
   delete_recur to delete and rebalance the tree
function delete(key, object_type with empty string as default) {
    if object_type is "person" then
        person = call search(key, "person")
        if person does not exist then
            display a messagage saying person does not exist
        else
            call delete_recur(key)
    else
         animal = call search(key)
         if animal does not exist then
            display a messagage saying animal does not exist
         else
            call delete_recur(root, animal_id)
```

```
            display message saying node has been deleted
}

function delete_recur(current_root, key) {
    if current_root is None then
        return current_root
    else if key < current_root's id then
        current_root's left branch = call delete_recur(current_root's
                                     left branch, key)
    else if key > current_root's id then
        current_root's right branch = call delete_recur(current_root's
                                      right branch, key)
    else if node is found then
        if node only has left child then
            temp = node's left child
            return temp
        else if node only has right child then
            temp = node's right child
            return temp
        else if node has 2 children then
            temp = call find_successor(current_root's right child)
            current_root's key = successor's key
            current_root's right child = call delete_recu(current_root's
                                         right child, successor's key)

    if curren_root is Node then
        return current_root

    update height of the curren_root node
    differece = get the height difference of the current_root node

    // check for balance and rotate if necessary
    if difference < -1 and height difference between current_root's
                          right child branches <= 0 then
        return rotate left around the current_root

    if difference > 1 and height difference between current_root's left
                        child branches >= 0 then
        return rotate left around the current_root

    if difference < -1 and height difference between current_root's
                          right child branches > 0 then
        rotate right around current_root
        return rotate left around the current_root

    if difference < -1 and height difference between current_root's
                          right child branches <= 0 then
        rotate left around the current_root
        return rotate right around the current_root

    return current_root
}
```

```
    // function to find the logical successor to the to be deleted node,
       which is the leftmost child of the node
    function find_successor(node){
        current = node
        while current's left child exists do
            current = current's left child
        return current
    }

    // function to traverse through the tree in alphabetical order and print
       animal infomation, it calls the recursive funtion in_order_recur
    function in_order(object_type with empty string as default){
        if object_type is "person" then
            call recursive function in_order_recur(root, "person)
        else
            call recursive function in_order_recur(root)
    }

    function in_order_recur(node, object_type with empty string as default) {
        if node is None then
            stops the recursion

        if object_type is "person" then
            call in_order_recur(node's left child, "person")
            display animal's information
            call in_order_recur(node's right child, "person")
        else
            call in_order_recur(for node's left child)
            display animal's information
            call in_order_recur(for node's right child)
    }

    // function to get height of a node
    function get_height(node) {
        if node exists then
            return node's height
        else
            return 0
    }

    // function to update height of a node
    function update_height(node) {
        node's height = the bigger of two node's children's heights + 1
    }

    // function to get height difference between node's left and right children
    function get_difference(node) {
        if node exists then
            return node's left child's height - node's right child's height
        else
            return 0
    }
}
```

IV. csv_to_tree.py

```
Pets tree          = new empty AVLTree
Wild Animals tree = new empty AVLTree
Treatments tree    = new empty AVLTree

open PETS.csv as file
    reader = file reader
    skip first line in file

    for every row in reader do
        sanctuary id         = row[0]
        animal type          = row[1]
        breed                = row[2]
        vaccinated           = row[3]
        neutered             = row[4]
        microchip            = row[5]
        reason for entry     = row[6]
        date of arrival      = row[7]
        date of departure    = row[8]
        destination          = row[9]
        destination address  = row[10]

        date of arrival = create an instance of Date for date of arrival
        date of departure = create an instance of Date for date of departure

        pet = create an instance of Pet with sanctuary id, animal type,
              vaccinated, reason for entry, date of arrival, date of
              departure, destination, destination address, breed, neutered,
              microchip
        insert pet into Pets tree


open WILD ANIMALS.csv as file
    reader = file reader
    skip first line in file

    for every row in reader do
        sanctuary id         = row[0]
        animal type          = row[1]
        vaccinated           = row[2]
        reason for entry     = row[3]
        date of arrival      = row[4]
        date of departure    = row[5]
        destination          = row[6]
        destination address = row[7]

        date of arrival = create an instance of Date for date of arrival
        date of departure = create an instance of Date for date of departure

        wild animal = create an instance of Wild Animal with sanctuary id,
              animal type, vaccinated, reason for entry, date of arrival,
              date of departure, destination, destination address
        insert wild animal into Wild Animals tree
```

```
open TREATMENTS.csv as file
     reader = file reader
     skip first line in file

     for every row in reader do
         sanctuary id       = row[0]
         surgery            = row[1]
         surgery date       = row[2]
         medication         = row[3]
         medication start              = row[4]
         medication finish             = row[5]
         responsible for abuse         = row[6]
         responsible for abandoning    = row[7]

         surgery date      = create an instance of Date for surgery date
         medication start  = create an instance of Date for medication start
         medication finish = create an instance of Date for medication finish

         treatment = create an instance of Pet with sanctuary id, surgery,
                     surgery date, medication, medication start, medication
                     finish, responsible for abuse, responsible for abandoning
         insert treatment into Treatments tree
```

V. Task 2

A. Create an entry for a new arrvial

```
function new_entry (category) {
    sanctuary id = get user input

    if user want to input entry on Pets or Wild Animals then
        animal type         = get user input
        vaccinated          = get user input
        reason for entry    = get user input
        date of arrival     = get user input, create a Date instance
        date of departure   = get user input, create a Date instance
        destination         = empty string
        destination address = empty string

        if date of departure is specified then
            destination = get user input
            destination address = get user input

        if user want to create new entry on Pets then
            breed     = get user input
            neutered  = get user input
            microchip = get user input

            pet = create a new Pet instance using inputted values
            insert pet into the Pet tree
            append the new pet entry into the Pets csv file

        else if user want to create new entry on Wild Animals then
            wild animal = create new Wild Animal object using inputted values
```

```
            insert wild animal into the Wild Animal tree
            append the new wild animal entry into the Wild Animals csv file

    else if user want to create new entry on Treatment then
        surgery           = get user input
        surgery date      = get user input and validate the date format
        medication        = get user input
        medication start  = get user input and validate the date format
        medication finish = get user input and validate the date format
        responsible for abuse      = get user input
        responsible for abandoning = get user input

        treatment = create a new Treatment entry using inputted value
        insert treatment into the Treatment tree
        append the new treatment entry to the Treatments csv file
}
```

B. Find the full data for an animal by using the animal's sanctuary identification

```
funtion find_animal (file_type, id) {
    if file_type is 'pet' then
        animal = call method 'search' in Pets tree
        return animal

    else if file_type is 'wild animal' then
        animal = call method 'search' in Wild Animals tree
        return animal

    else if file_type is 'treatment' then
        animal = call method 'search' in Treatments tree
        return animal
}
```

C. List of people that have abused/abandoned their animals in alphabetical order

```
function abused_abandoned_list (crime) {
    people_crime = new AVLTree

    // function to recursively find abusers or abandoners and add them to
       the list
    funtion in_order_recur(node) {
        if node does not exist then
            stop recursion
        call function in_order_recur(left node of node)
        if crime is 'abuse' then
            person = responsible_for_abuse of animal
        else
            person = responsible_for_abandoning of animal
        if person is not empty and person is not already in people_crime then
            insert person into people_crime
    }

    call function in_order_recur (root node of Treatments tree)
    return people_crime
}
```

D. List of Cats/Dogs ready for adoption

```
function pets_adoption (pet_type) {
    adopt_ready = empty AVLTree

    // function to recursively find cats or dogs and add them to adopt_ready
    function in_order_recur (node) {
        if node does not exist then
            stop recursion
        call function in_order_recur (left node of current node)
        if animal_type of current node's animal is pet_type
               and animal is vaccinated
               and animal is neutered
               and animal is microchipped then
            insert current node's animal in the adopt_ready tree
        call function in_order_recur (right node of current node)
    }
    call function in_order_recur(root node of Pets tree)
    return adopt_ready
}
```

E. List of Pets, Wild Animals that are ready to be returned to owners in ascending order of sanctuary ID, with pets listed first

```
function return_to_owner() {
    return_ready = new empty AVLTree

    // function to go through the Pets and Wild Animals Tree and check if
       animal is ready to be returned to owner
    function check_ready(current_root) {
        if current_root does not exist then
            stop recursion
        call check_ready(current_root's left child)

        animal = current_root's animal
        if animal is vaccinated and animal has not left sanctuary then
            treatment = search for animal in Treatments tree
            if (animal is not in Treatments tree) or (animal is in Treatments
            tree and animal's medication finish date is specified) then
                 if (animal is of type Pet and animal's reason for admission
                 is not 'Abused' or 'Abandoned') or (animal is of type Wild
                 Animal) then
                     insert animal into return_ready tree
        call check_ready(current_root's right child)
    }

    call check_ready(Pets tree's root)
    call check_ready(Wild Animals tree's root
    return return_ready tree
}
```

VI. Task 3

A. Enter Details of Surgery

```
function enter_surgery(animal_id) {
    animal = search for animal in Treatments tree
    if animal is not found then then
        display a message saying animal is not found
        exit function

    if animal's surgery has already been specified then
        display a message saying animal's surgery has already been specified
                and user cannot change it
    else
        while True do
             surgery      = get user input
             surgery date = get user input and create a Date instance
             if surgery and surgery date are not empty string then
                  break

        assign surgery to animal's surgery
        assign surgery date to animal's surgery date
        write the treatments tree with edited data into the TREATMENTS.csv
}
```

B. Enter details of neutering

```
function enter_neutering(animal_id) {
   animal = search for animal_id in Pets tree
   if animal is not found then
        display a message saying animal is not found
        exit function

   if animal's neutered is 'Yes' then
        display a message saying animal is neutered and user cannot change it
   else
        animal's neutering = 'Yes'
        write the Pets tree with edited neutering detail into PETS.csv
        diplay a message saying animal's neutering is updated
}
```

C. Enter Pet's microchip number

```
function edit_microchip(animal_id) {
   animal = search for animal_id in Pets tree
   if animal is not found then
        display a message saying animal is not found
        exit function

   if animal's microchip is already specified then
        display a message saying animal is microchipped and user cannot
        change it
    else
         detail = get user input
         assign detail to animal's microchip
```

```
        write the Pets tree with edited data into PETS.csv
        display the updated microchip details of animal
}
```

D. Edit date of departure from sanctuary

```
function edit_depart_date(animal_id) {
    animal = search for animal in Pets tree, Wild Animals tree
    if aninal is not found then
        display a message saying animal is not found
        exit function

    display the current details of animal's departure date
    date = get user input and create an instance of Date
    assign date to animal's date of departure

    if date is not specified then
        animal's destination         = empty string
        animal's destination address = empty string
    else
        if both animal's destination and destination address have not already
           been specified then
            while True do
                destination         = get user input
                destination address = get user input
                if destination and destination address are not empty then
                    break
            assign destination to animal's destination
            assign destination address to animal's destination address


    if animal is of type Pet then
    write Pets tree with edited details into PETS.csv
    else if animal is of type WildAnimal then
    write Wild Animals tree with edited details into WILD ANIMALS.csv

    display updated state of animal's date of departure
    display updated state of animal's destination and destination address
}
```

E. Edit Destination of animal upon departure

```
function edit_dest(animal_id) {
    animal = search for animal_id in Pets tree and Wild Animals tree
    if animal is not found then
        display a message saying animal is not found
        exit function

    if animal's departure date has not been specified then
        display a message saying departure date must be specified first
        call function edit_depart_date(animal_id)
        exit function

    display the current animal's destination and destination address
```

```
    destination = get user input
    assign destination to animal's destination

    if destination is specified then
        while True do
            destination_address = get user input
            if destination_address is not blank then
                break
        assign destination_address to animal's destination address
    else
        animal's date of departure will also be blank
        animal's destination address will also be blank

    if animal is of type Pet then
        write Pets tree with edited details into PETS.csv
    else if animal is of type WildAnimal then
        write Wild Animals tree with edited details into WILD ANIMALS.csv

    display the updated animal's destination and destination address
}
```

VII. Other Functions

A. Search for animal ID in Pets tree and Wild Animals tree

```
function check_ID(animal_id) {
    if first letter of animal_id is 'P' then
        animal = search for animal_id in Pets tree
        return animal
    else if first letter of animal_id is 'W' then
        animal = search for animal_id in Wild Animals tree
        return animal
    else
        return False
}
```

B. Return all data of an animal in Pets or Wild Animals or Treatments trees

```
function file_attributes (file_type, animal) {
    if file_type is 'pet' then
        return a list with [sanctuary id, type, breed, vaccinated, neutered,
            microchip, reason for entry, date of arrival, date of
            departure, destination, destination address] of animal
    else if file_type is 'wild animal' then
        return a list with [sanctuary id, type, vaccinated, reason for
            entry, date of arrival, date of departure, destination,
            destination address] of animal
    else if file_type is 'treatment' then
        return a list with [sanctuary id, surgery, surgery date, medication,
            medication start, medication finish, responsible for abuse,
            responsible for abandoning] of animal
}
```

C. Write Pets, Wild Animals and Treatments tree into their csv files

```
function write_to_file(file_type) {
    function in_order_recur(node) {
        if node does not exist then
            stop recursion
        in_order_recur(node's left child)
        write the current node's animal into file_type
        in_order_recur(node's right child)
    }

    if file_type is 'pet' then
        file = PETS.csv
    else if file_type is 'wild animal' then
        file = WILD ANIMALs.csv
    else if file_type is 'treatment; then
        file = TREATMENTS.csv

    open file in write mode
        if file_type == 'pet'then
            write Pets headings into file
            recursively write animals of Pets tree into PET.csv file

        else if file_type == 'wild animal' then
            write Wild Animals headings into file
            recursively write animals of Wild Animals tree into
            WILD ANIMAL.csv file

        else if file_type == 'treatment' then
            write Treatment headings into file
            recursively write animals of Treatments tree into TREATMENT.csv
            file
}
```

D. Display full data of a tree

```
function print_full(file_type) {
    if file_type is 'pet' then
        return print Pets tree in_order
    else if file_type is 'wild animal' then
        return print Wild Animals tree in_order
    else if file_type is 'treatment' then
        return print Treatments tree in_order
}
```