CS2021 Practice Final Exam

You should be prepared to answer questions that relate closely to the labs and homework assignments and following readings: Composing Programs (Chapters 1,2, 4.2), Course Notes on BB

Topics from before Midterm
Functions and Expressions
Control Structures
Higher Order Functions
Lambdas
Environment diagrams
Recursion
Tree Recursion
Lists, Dictionaries, and Sequences

Topics from after Midterm
Data Abstraction
Linked Lists (ADT)
Mutable objects and functions
Modules
Object-Oriented Programming
Inheritance
Mutable Linked Lists (OOP)
Interfaces
Iterators and generators

Questions 1-17 Select True/False

1. T/F A pair can be constructed using list objects.
2. T/F A sequence must support element selection.

3. **T/F** List Comprehensions enable many sequence processing operations expressed by evaluating a fixed expression.

4. T/F The builtin map and filter operations are functions that apply to any sequence.

5. T/F A linked list can be implemented with any pair construction.

6. T/F Two lists may have the same contents but in fact be different lists, which can be tested with comparison operator called is.

7. T/F A dictionary contains key-value pairs, where both the keys and values are objects.

8. **T/F** The "state" of a python execution implies an evolving process in which values of identities may change over time.

9. T/F Lists and dictionaries do not have *state*, and their values cannot be changed over time.

10. T/F The method that initializes objects has a special name __init__

11. T/F Every object that is an instance of a user-defined class has a unique identity.

12. T/F An attribute of a class and an attribute of an object are the same thing.

13. T/F A method is just a function, but as an attribute of an instance, it is called a bound method.

14. T/F An *interface* is a set of shared attribute names, along with a specification of their behavior, which can be implemented in different classes.

15. T/F An iterator is an object that has a next method.

16. T/F A generator function is iterable.

17. T/F A generator function when called returns an iterator.

Questions 18-32. What Would Python Print?
In each of the following, provide an exact representation of what python would print using interactive commands.

18.
```
>>> yo = [1]
>>> len(yo)
```
_____
19.
```
>>> 1 in yo
```
_____
20.
```
>>> yo + [2, 3]
```
_____
21.
```
>>> [0] + yo
```
_____
22.
```
>>> yo * 3
```
_____
23.
```
>>> zoo = [[1, 2], [3, 4, 5]]
>>> len(zoo)
```
_____
24.
```
>>> list(map(lambda x: x*x, [1, 2, 3, 4]))
```
_____

25.
```
>>> list(filter(lambda x: x % 2 == 0, [1, 2, 3, 4]))
```
_____

26.
```
>>> seq = [1, 2, 3, 4]
>>> [x * x for x in seq if x % 2 == 0]
```
27.
```
>>> [x*x for x in range(5)]
```
_____

28.
```
>>> [n for n in range(10) if n % 2 == 0]
```
_____

29.
```
>>> ones = [1 for i in ["hi", "bye", "you"]]
>>> ones + [str(i) for i in [1, 2, 3]]
```
_____

30.
```
>>> [i+5 for i in [n for n in range(1,4)]]
```
_____

31.
```
>>> type(2)
```
_____

32.
```
>>> type(1.5)
```
_____

Questions 18-32
Provide the correct Python Command to produce the associated output.
33.
```
>>> xo = [1, 3, [5, 7], 9]
>>> _____________
7
```

34.
```
>>> xo = [[7]]
>>> _____________
7
```

35.
```
>>> xo = [1, [2, [3, [4, [5, [6, [7]]]]]]]
>>> _____________
7
```

.
36.
```
>>> pokemon = {'pikachu': 25, 'dragonair': 148, 'mew': 151}
>>> _____________
25
```

36.
```
>>> pokemon['jolteon'] = pokemon[‘mew’]
>>> _____
151
```

Questions 37-44 uses the following Linked List definition.

```
empty = 'empty'

def is_link(s):
    """s is a linked list if it is empty or a (first, rest) pair."""
    return s == empty or (type(s) == list and len(s) == 2 and is_link(s[1]))

def link(first, rest=empty):
    """Construct a linked list from its first element and the rest."""
    assert is_link(rest), 'rest must be a linked list.'
    return [first, rest]

def first(s):
    """Return the first element of a linked list s."""
    assert is_link(s), 'first only applies to linked lists.'
    assert s != empty, 'empty linked list has no first element.'
    return s[0]

def rest(s):
    """Return the rest of the elements of a linked list s."""
    assert is_link(s), 'rest only applies to linked lists.'
    assert s != empty, 'empty linked list has no rest.'
    return s[1]
```

37.
```
>>> one = link(1)
>>> first(one)
```
_____

38.
```
>>> rest(one)
```
_____

39.
```
>>> is_link(one)
```
_____

40.
```
>>> is_link(first(one))
```
_____

41.
```
>>> is_link(rest(one))
```
_____

42.
```
>>> four = link(1, link(2, link(3, link(4, empty))))
>>> first(four)
```
_____

43..
```
>>> rest(four)
```
_____

44.
```
>>> is_link(rest(four))
```
_____

45. Consider the following simple class def.
```
class Dog(object):
    def bark(self):
        print("woof!")
```

One day, while using this class, Louis Reasoner decides he wants his dog, Fido, to bark differently:

```
>>> fido = Dog()
>>> fido.bark = "bow wow!"
```

Ben Bitdiddle quickly points out that this won't work. "bark is supposed to be a method, not a string!"
So Louis Reasoner attempts to reset the bark method to what it was before:
```
>>> fido.bark = Dog.bark
```
Ben replies, "I don't think your fix is right either!"
Circle all appropriate statements about this final assignment statement.
(A) Executing this assignment statement will cause an error.
(B) After this assignment, invoking fido.bark() will cause an error.
(C) This assignment statement will have no effect at all.
(D) None of the above criticisms are valid

46. The Person class is defined as follows:
```
class Person(object):
  name = None
  def __init__(self, name):
    Person.name = name
  def greet(self):
   return "Hello, my name is " + self.name
```

Circle all appropriate criticisms of this implementation.
(A) Every Person's name will be the equal to the most recently created Person's name.
(B) Instantiating a Person will cause an error.
(C) Every Person's name will be None.
(D) Invoking greet on a person instance will cause an error.

Questions 47-53.
The Owner and Pet classes are defined as follows:

```
class Owner(object):
    all = []

    def __init__(self, name):
        self.name = name
        self.pets = []
        Owner.all.append(self)

    def add_pet(self, pet):
        self.pets.append(pet)

    def __repr__(self):
        return 'Owner(' + self.name + ')'

class Pet(object):
    all = []

    def __init__(self, name, weight, height):
        self.name = name
        self.weight = weight
        self.height = height
        Pet.all.append(self)

    def bmi(self):
        return self.weight / (self.height * self.height)

    def __repr__(self):
        return 'Pet(' + self.name + ')'

>>> bob = Owner('bob')
>>> joe = Owner('joe')
>>> bob.all                #45

>>> bob.all.append(bob)
>>> joe.all                #46

>>> type(joe.add_pet)       #47

>>> type(Owner.add_pet)     #48

>>> harry = Pet('harry', 50, 50)
>>> type(harry.bmi)         #49

>>> joe.pets.append(harry)
>>> bob.add_pet(harry)
>>> bob.pets[0].all         #50

>>> bob.pets.append(Pet('jimmy', 40, 10))
>>> bob.pets[1].owner       #51

>>> Pet.all                 #52

>>> jimmy                   #53
```