CS320 HW5 Jiacheng Zhao

QUESTION 1:
a)
Step 11 of the assignment asks for an implementation of a method that will produce a string that describes the elements of two lists in a pairwise fashion:
For example when give 2 and 4 it will become (2, 4)

It is easy to do this for a Nil list: just return self:

```
  def zip(self, other):   # in the Nil class
    return self
```

What code should we use in the Cons class? The problem here is that we need to decide which number should be put first and which number will be second (the order), and to make the decision, we need to know whether the tail of list is empty or not. In pseudo code, this might look something like the following:

```
  def zip(self, other):   # in the Cons class
    if other.tail is an empty list:
      return self
    else:
      return Cons( (other.head, self.head) , other.tail.zip(self.tail))
```

The "other.tail an empty list" part of this code is not valid code, but we can eliminate it by using the dynamic dispatch pattern shown above. The first step is to rewrite the definition of commaElements in the Cons class (we'll figure out what to use in place of "..." shortly, but for now it's just a placeholder):

```
  def zip(self, other):   # in the Cons class
    return other.zipElement(...)
```

Next we add implementations of the new zipElement method in each of the Nil and Cons classes:

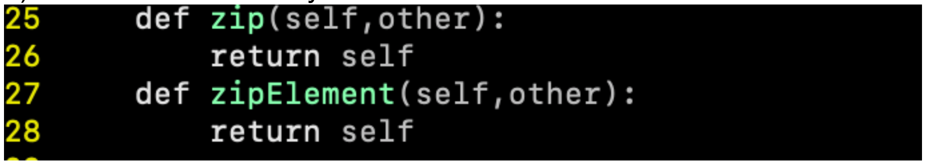
```
  def zipElement(self, ...):   # in the Nil class
    return self

  def zipElement(self, ...):   # in the Cons class
    return Cons( (other.head, self.head) , other.tail.zip(self.tail))
```

Now we can see that both of these methods require "head" as an input, so we can pass that value as a parameter in the calls to zipElement:

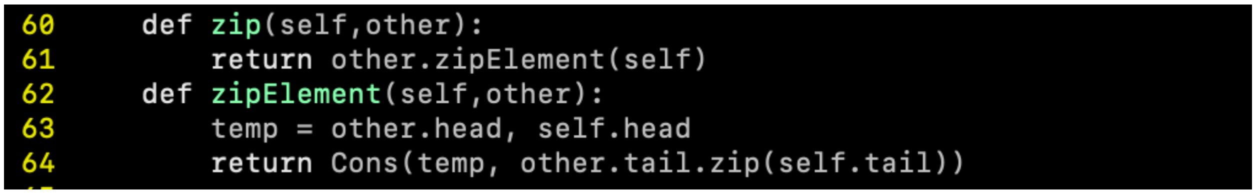
```
  def zip(self, other):   # in the Cons class
    return other.zipElement(self)

  def zipElement(self, other):     # in the Nil class
    return self

  def zipElement(self, other):     # in the Cons class
    temp = other.head, self.head
    return Cons(temp, other.tail.zip(self. Tail))
```

Together, those three method definitions complete the implementation of zip().

b) This is screenshot my code:



```
    def zip(self,other):
        return self
    def zipElement(self,other):
        return self
```



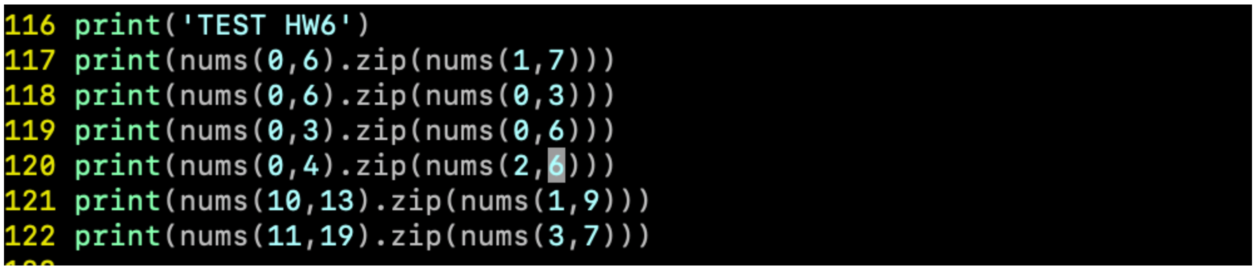
```
    def zip(self,other):
        return other.zipElement(self)
    def zipElement(self,other):
        temp = other.head, self.head
        return Cons(temp, other.tail.zip(self.tail))
```

c) Thoughtful Test: I have created three more test for code.

First is num(0,4).zip(num(2,6)). The reason I choose this is to try function part get same number with object part which 4 in (0,4) of function part is same 4 in (2,6) of object part.
The output I except is 4 parts, because (0,4) only include 4 numbers which same with (2,6). The number will begin at 0 and end at 4. Begin at 2 end at 5.
It will come out (0,2) (1,3) (2,4) (3,5)

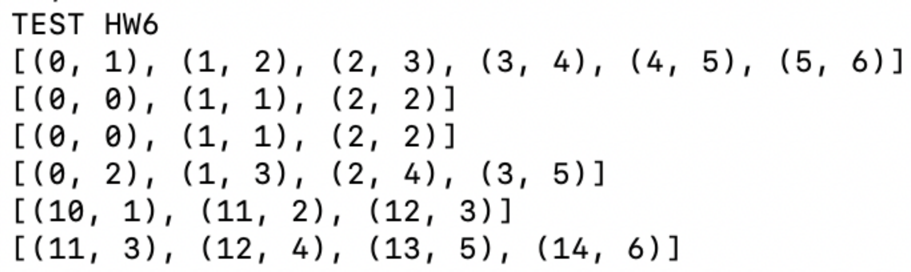
Second is num(10,13).zip(num(1,9)). The reason I choose this is to try function part get more number than object part which 8 in (1,9) of function part is more than 3 in (10,13) of object part. The output I except is only 3 parts, because (10,13) only include 3 numbers which smaller than (1,9). The number will begin at 10 and end at 12. Begin at 1 end at 3.
It will come out (10, 1) (11,2) (12,3)

Third is num(11,19).zip(num(3,7)). The reason I choose this is to try function part get smaller number than object part which 4 in (3,7) of function part is smaller than 8 in (11,19) of object part. The output I except is only 4 parts, because (3,7) only include 4 numbers which smaller than (11,19) the number will begin at 11 and end at 14. Begin at 3 end at 6.
It will come out (11, 3) (12,4) (13,5) (14,6)



```
print('TEST HW6')
print(nums(0,6).zip(nums(1,7)))
print(nums(0,6).zip(nums(0,3)))
print(nums(0,3).zip(nums(0,6)))
print(nums(0,4).zip(nums(2,6)))
print(nums(10,13).zip(nums(1,9)))
print(nums(11,19).zip(nums(3,7)))
```

I get same result which I except.



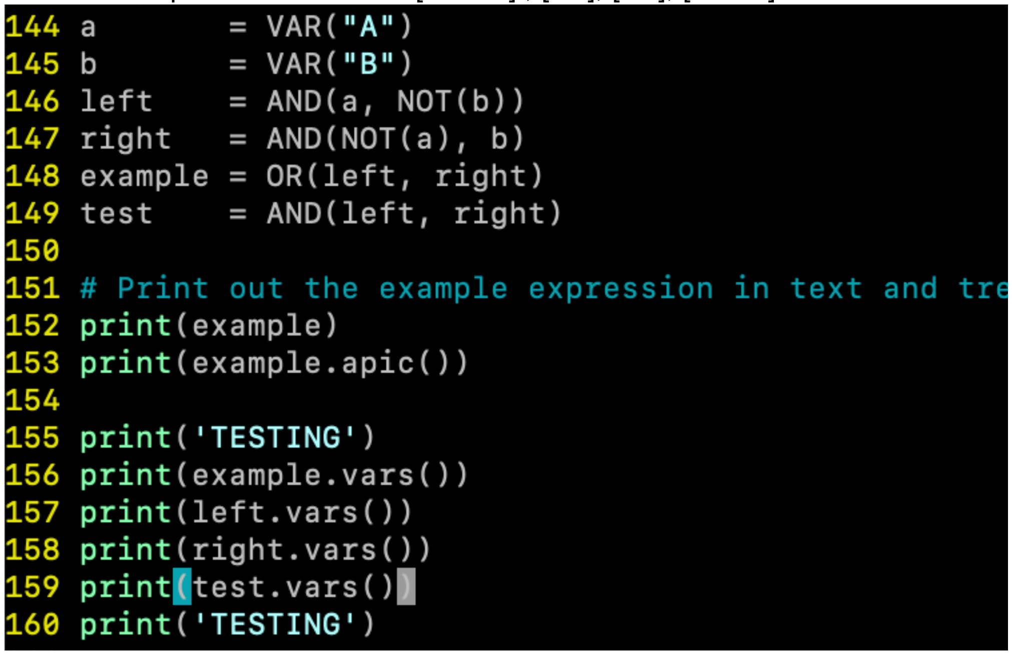
```
TEST HW6
[(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6)]
[(0, 0), (1, 1), (2, 2)]
[(0, 0), (1, 1), (2, 2)]
[(0, 2), (1, 3), (2, 4), (3, 5)]
[(10, 1), (11, 2), (12, 3)]
[(11, 3), (12, 4), (13, 5), (14, 6)]
```

QUESTION 2:

a) The vars functions return the list of variables in an arbitrary Prop value.

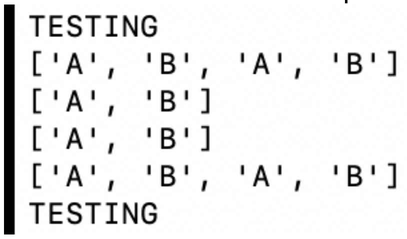
```
TRUE class:
        def vars(self):
            return [ ]
FALSE class:
        def vars(self):
            return [ ]
VAR class:
        def vars(self):
            return [self.name]
AND class:
        def vars(self):
            return self.p.vars( ) + self.q.vars( )
OR class:
        def vars(self):
            return self.p.vars( ) + self.q.vars( )
NOT class:
        def vars(self):
            return self.p.vars( )
```

Test: add three more test which listed above.
I try to test all 6 vars functions I implemented in 6 classes.
The result expected should be: [ABAB] , [AB], [AB], [ABAB]



```
a       = VAR("A")
b       = VAR("B")
left    = AND(a, NOT(b))
right   = AND(NOT(a), b)
example = OR(left, right)
test    = AND(left, right)

# Print out the example expression in text and tre
print(example)
print(example.apic())

print('TESTING')
print(example.vars())
print(left.vars())
print(right.vars())
print(test.vars())
print('TESTING')
```

The result is same with expected it shows below.

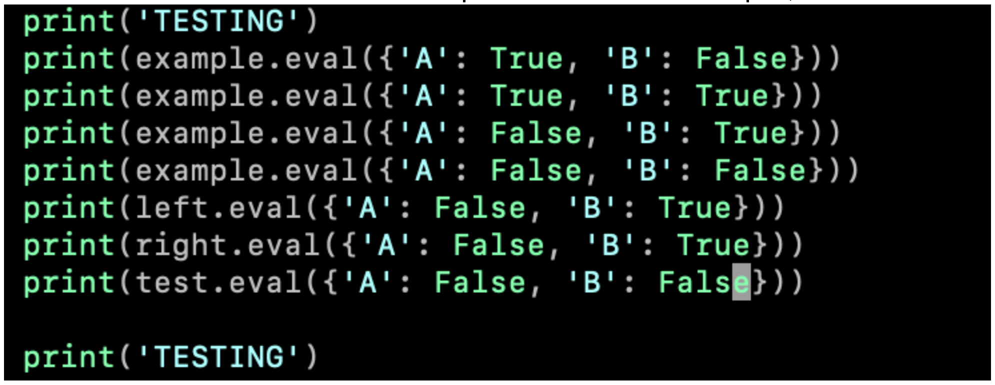


```
TESTING
['A', 'B', 'A', 'B']
['A', 'B']
['A', 'B']
['A', 'B', 'A', 'B']
TESTING
```

b)

```
TRUE class:
        def eval(self, env):
            return true
FALSE class:
        def eval(self, env):
            return false
VAR class:
        def eval(self, env):
            return env.get(self.name)
AND class:
        def eval(self, env):
            return self.p.eval(env) and self.q.eval(env)
OR class:
        def eval(self, env):
            return self.p.eval(env) or self.q.eval(env)
NOT class:
        def eval(self, env):
            return not self.p.eval(env)
```

Test: add six more test: True and True, False and True, False and False.
Expectation or these 4: True, False, True, False, False, True, False
Left = AND(False, Not True) -> False . right = AND(NOT False, True) -> True
test have different result with example when two False input, which will come out False.



```
print('TESTING')
print(example.eval({'A': True, 'B': False}))
print(example.eval({'A': True, 'B': True}))
print(example.eval({'A': False, 'B': True}))
print(example.eval({'A': False, 'B': False}))
print(left.eval({'A': False, 'B': True}))
print(right.eval({'A': False, 'B': True}))
print(test.eval({'A': False, 'B': False}))

print('TESTING')
```

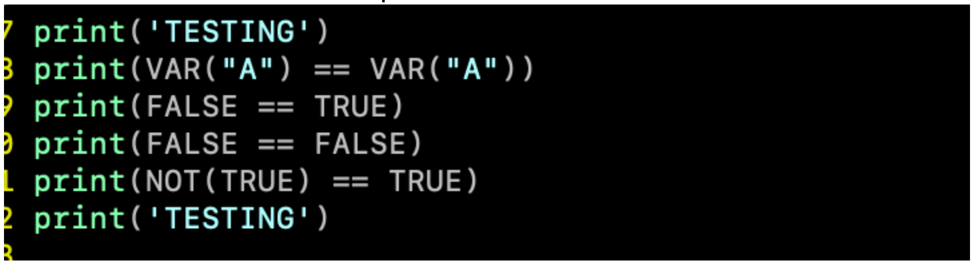
The result is same with expected it shows below.

```
TESTING
True
False
True
False
False
True
False
TESTING
```

c)

```
TRUE class:
        def eqTRUE(self):
          return True
        def __eq__(self,other):
          return other.eqTRUE( )
FALSE class:
        def eqFALSE(self):
          return True
        def __eq__(self,other):
          return other.eqFALSE(self)
VAR class:
        def eqVAR(self,other):
          return True
        def __eq__(self,other):
          return other.eqVAR(self)
AND class:
        def eqAND(self):
          return True
        def __eq__(self,other):
          return other.eqAND(self)
OR class:
        def eqOR(self):
          return False
        def __eq__(self,other):
          return other.eqOR(self)
NOT class:
        def eqNOT(self):
          return True
        def __eq__(self,other):
          return other.eqNOT(self)
```

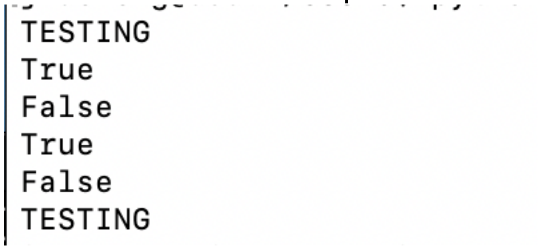
Test: add three more test. Expect it will come out: True False True False



```
print('TESTING')
print(VAR("A") == VAR("A"))
print(FALSE == TRUE)
print(FALSE == FALSE)
print(NOT(TRUE) == TRUE)
print('TESTING')
```

My TURE and FALSE part is ok. But others no works good.
If want to test, you can just implement TURE, FALSE and NOT part.

The result is same with expected it shows below.



```
TESTING
True
False
True
False
TESTING
```