Started on	Friday, 17 February 2023, 1:25 PM
State	Finished
Completed on	Friday, 17 February 2023, 3:25 PM
Time taken	2 hours
Marks	0.00/17.00
Grade	0.00 out of 10.00 (0 %)

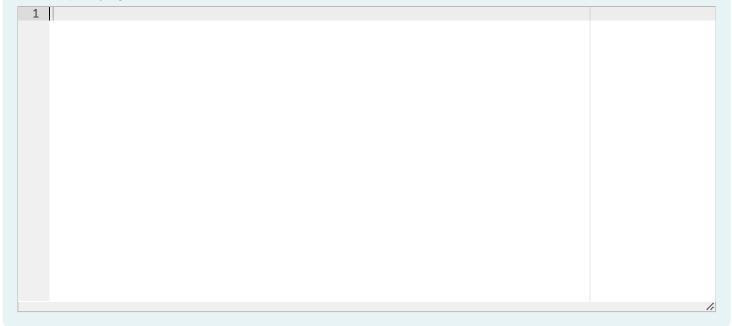
Not answered

Mark 0.00 out of 1.00

Use recursive approach to write a function lstSquare(n:Int) that returns a list of the squares of the numbers from 1 to n?

For example:

Test	Result
lstSquare(3)	[1,4,9]



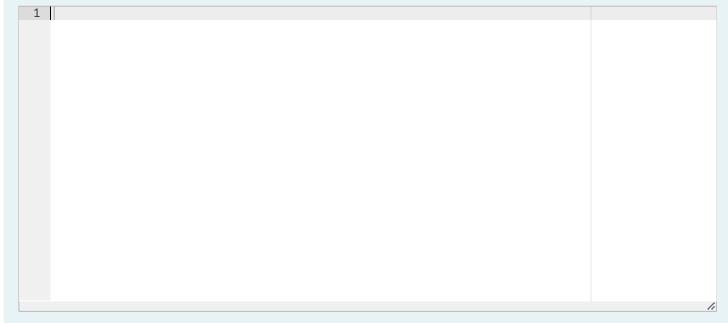


Mark 0.00 out of 1.00

Use list comprehension approach to write a function lstSquare(n:Int) that returns a list of the squares of the numbers from 1 to n?

For example:

Test	Result
lstSquare(3)	[1,4,9]



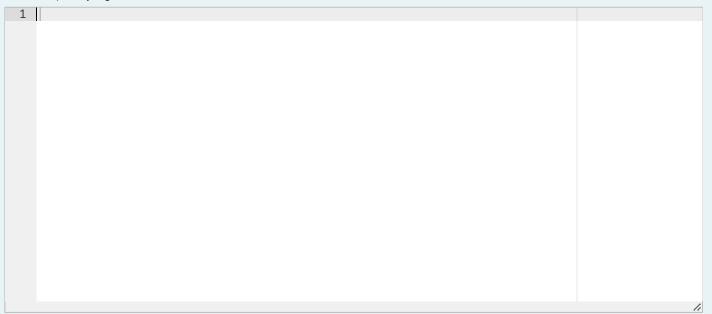


Mark 0.00 out of 1.00

Use high-order function approach to write function lstSquare(n:Int) to return a list of i square for i from 1 to n?

For example:

Test	Result
lstSquare(3)	[1,4,9]



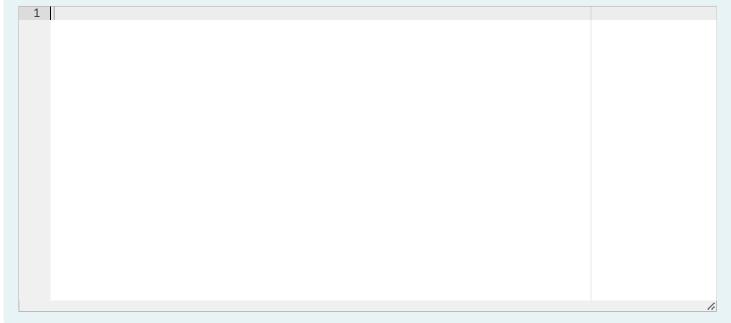


Mark 0.00 out of 1.00

Let lst be a list of integer and n be any value, use **recursive approach** to write function dist(lst,n) that returns the list of pairs of an element of lst and n.

For example:

Test	Result
dist([1,2,3],4)	[(1, 4),(2, 4),(3, 4)]





Mark 0.00 out of 1.00

Let lst be a list of integer and n be any value, use list comprehension approach to write function dist(lst,n) that returns the list of pairs of an element of lst and n.

For example:

Test	Result
dist([1,2,3],4)	[(1, 4),(2, 4),(3, 4)]



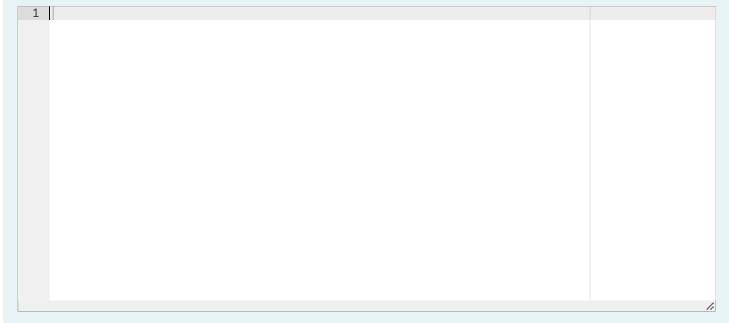


Mark 0.00 out of 1.00

Let lst be a list of integer and n be any value, use **high-order function approach** to write function dist(lst,n) that returns the list of pairs of an element of lst and n.

For example:

Test	Result
dist([1,2,3],4)	[(1, 4),(2, 4),(3, 4)]



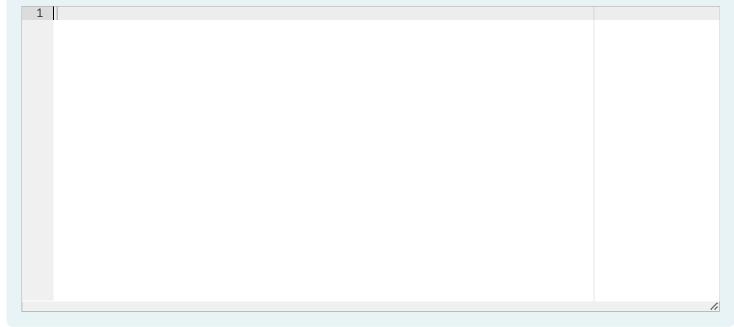
Not answered

Mark 0.00 out of 1.00

Let lst be a list of integer and n be an integer, use **recursive approach** to write function lessThan(lst,n) that returns the list of all numbers in lst less than n.

For example:

Test	Result
lessThan([1,2,3,4,5],4)	[1,2,3]



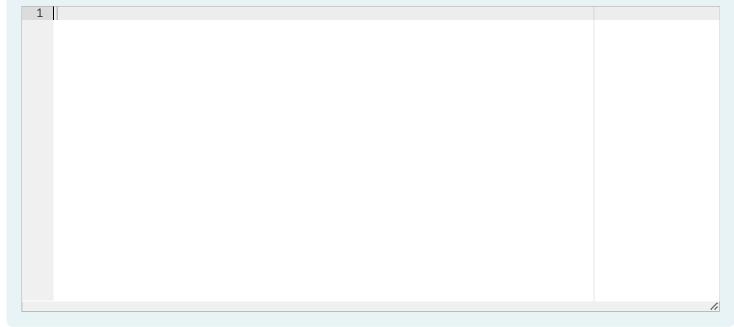


Mark 0.00 out of 1.00

Let lst be a list of integer and n be an integer, use list comprehension approach to write function lessThan(lst,n) that returns the list of all numbers in lst less than n.

For example:

Test	Result
lessThan([1,2,3,4,5],4)	[1,2,3]



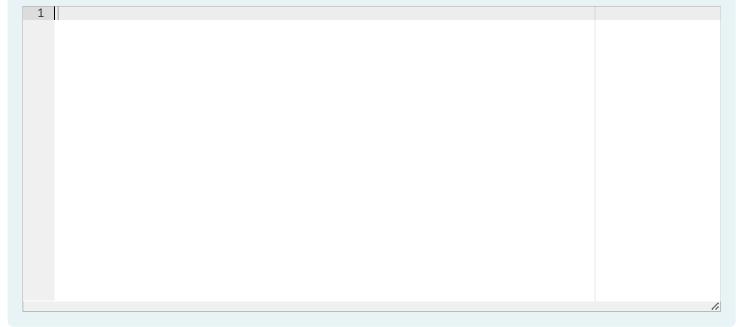


Mark 0.00 out of 1.00

Let lst be a list of integer and n be an integer, use **high-order function approach** to write function lessThan(lst,n) that returns the list of all numbers in lst less than n.

For example:

Test	Result
lessThan([1,2,3,4,5],4)	[1,2,3]



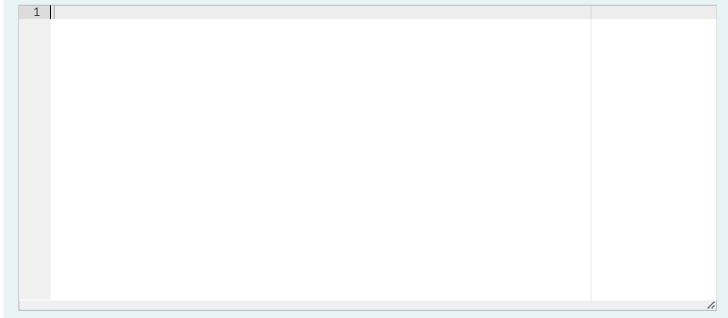
Not answered

Mark 0.00 out of 1.00

Let lst be a list of a list of element, use recursive approach to write function flatten(lst) that returns the list of all elements

For example:

Test	Result
flatten([[1,2,3],[4,5],[6,7]])	[1,2,3,4,5,6,7]



Not answered

Mark 0.00 out of 1.00

Let lst be a list of a list of element, use list comprehension approach to write function flatten(lst) that returns the list of all elements

For example:

Test	Result
flatten([[1,2,3],[4,5],[6,7]])	[1,2,3,4,5,6,7]



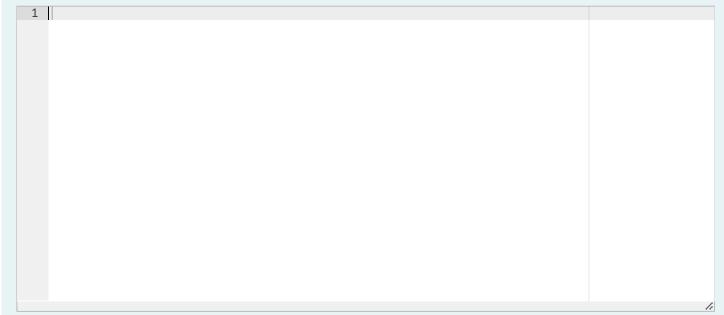
Not answered

Mark 0.00 out of 1.00

Let lst be a list of a list of element, use high-order function approach to write function flatten(lst) that returns the list of all elements

For example:

Test	Result	
flatten([[1,2,3],[4,5],[6,7]])	[1,2,3,4,5,6,7]	



Not answered

Mark 0.00 out of 1.00

To express an arithmetic expression, there are 5 following classes:

Exp: general arithmetic expression

BinExp: an arithmetic expression that contains one binary operators (+,-,*,/) and two operands. To construct a BinExp object, you must pass parameters: first operand, operator, second operand, respectively.

UnExp: an arithmetic expression that contains one unary operator (+,-) and one operand. To construct a UnExp object, you must pass the operator first.

IntLit: an arithmetic expression that contains one integer number

FloatLit: an arithmetic expression that contains one floating point number

Define these classes in Python (their parents, attributes, methods) such that their objects can response to eval() message by returning the value of the expression. For example, let object x express the arithmetic expression 3 + 4 * 2.0, x.eval() must return 11.0

In this exercise, we use:

```
x1 = IntLit(1)
```

x2 = FloatLit(2.0)

x3 = BinExp(x1,"+",x1)

x4 = UnExp("-",x1)

x5 = BinExp(x4,"+",BinExp(IntLit(4),"*",x2))

For example:

Test	Result
<pre>print(x1.eval())</pre>	1



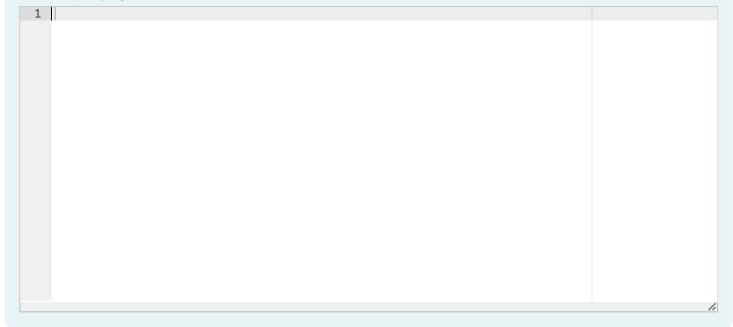
Not answered

Mark 0.00 out of 1.00

Extend the contents of classes Exp, BinExp, UnExp, IntLit, FloatLit such that they can response to printPrefix() message to return the string corresponding to the expression in prefix format. Note that, unary operator +/- is printed as +./-. in prefix format and there is a space after each operator or operand. For example, when receiving message printPrefix(), the object expressing the expression -4 + 3 * 2 will return the string "+ - .4 * 3 2"

For example:

Test	Result
<pre>print(x1.printPrefix())</pre>	1



Not answered

Mark 0.00 out of 1.00

As in the previous question, when a task is added into expression classes, new methods are added into these classes. Please change the way these classes are implemented in such a way that these classes do not change their contents when new tasks are added into these classes:

- Define class Eval to calculate the value of an expression.
- Define class PrintPrefix to return the string corresponding to the expression in prefix format.

All arithmetic classes in previous questions have just been defined as follows:

```
class Exp(ABC):pass
class BinExp(Exp):
    def __init__(self,o1,op,o2):
        self.left = o1
        self.op = op
        self.right = o2
class UnExp(Exp):
    def __init__(self,op,o1):
        self.op = op
        self.operand = o1
class IntLit(Exp):
    def __init__(self,v):
        self.value = v
class FloatLit(Exp):
    def __init__(self,v):
        self.value = v
```

Let v1, v2 be an object of Eval, PrintPrefix and x be an object expressing an expression, v1.visit(x1) will return the value of the expression x and v2.visit(x) will return the expression in prefix format.

For testing, given some following objects:

```
x1 = IntLit(1)

x2 = FloatLit(2.0)

x3 = BinExp(x1,"+",x1)

x4 = UnExp("-",x1)

x5 = BinExp(x4,"+",BinExp(IntLit(4),"*",x2))
```

Hint: use type(), isinstance() to find out the type of x when implementing this exercise.

For example:

Test	Result
<pre>print(v1.visit(x1))</pre>	1
<pre>print(v2.visit(x1))</pre>	1

Answer: (penalty regime: 10, 20, ... %)

Reset answer

```
1 | class Eval: pass
2 | class PrintPrefix: pass
3 | class PrintPostfix: pass
4 |
```

Not answered

Mark 0.00 out of 1.00

As in the previous question, when a task is added into expression classes, new methods are added into these classes. Please change the way these classes are implemented in such a way that these classes do not change their contents when new tasks are added into these classes:

- Define class Eval to calculate the value of an expression.
- Define class PrintPrefix to return the string corresponding to the expression in prefix format.

All arithmetic classes in previous questions have just been defined as follows:

```
class Exp(ABC):pass
class BinExp(Exp):
    def __init__(self,o1,op,o2):
        self.left = o1
        self.op = op
        self.right = o2
    def accept(self,v): return v.visitBinExp(self)
class UnExp(Exp):
    def __init__(self,op,o1):
        self.op = op
        self.operand = o1
    def accept(self,v): return v.visitUnExp(self)
class IntLit(Exp):
    def __init__(self,v):
        self.value = v
    def accept(self,v): return v.visitIntLit(self)
class FloatLit(Exp):
    def __init__(self,v):
        self.value = v
    def accept(self,v): return v.visitFloatLit(self)
```

Let v1, v2 be an object of Eval, PrintPrefix and x be an object expressing an expression, v1.visit(x1) will return the value of the expression x and v2.visit(x) will return the expression in prefix format.

For testing, given some following objects:

```
x1 = IntLit(1)
x2 = FloatLit(2.0)
x3 = BinExp(x1,"+",x1)
x4 = UnExp("-",x1)
x5 = BinExp(x4,"+",BinExp(IntLit(4),"*",x2))
```

Be careful that you should not allowed to use type(), isinstance() when implementing this exercise.

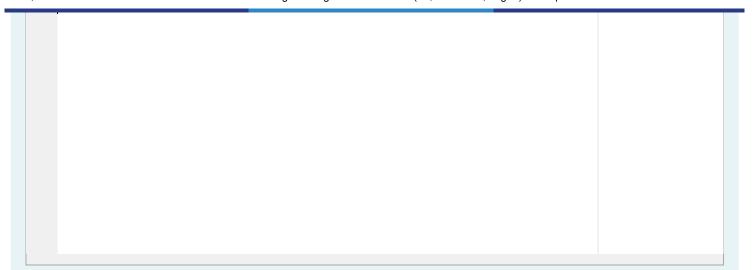
Tip: Use Visitor pattern.

For example:

Test	Result
<pre>print(v1.visit(x1))</pre>	1
<pre>print(v2.visit(x1))</pre>	1

```
Reset answer
```

```
1 | class Eval: pass
2 | class PrintPrefix: pass
3 | class PrintPostfix: pass
```



Not answered

Mark 0.00 out of 1.00

As in the previous question, when a task is added into expression classes, new methods are added into these classes. Please change the way these classes are implemented in such a way that these classes do not change their contents when new tasks are added into these classes:

- Define class Eval to calculate the value of an expression.
- Define class PrintPrefix to return the string corresponding to the expression in prefix format.
- Define class PrintPostfix to return the string corresponding to the expression in postfix format.

All arithmetic classes in previous questions have just been defined as follows:

```
class Exp(ABC):pass
class BinExp(Exp):
    def __init__(self,o1,op,o2):
        self.left = o1
        self.op = op
        self.right = o2
    def accept(self,v): return v.visitBinExp(self)
class UnExp(Exp):
    def __init__(self,op,o1):
        self.op = op
        self.operand = o1
    def accept(self,v): return v.visitUnExp(self)
class IntLit(Exp):
    def __init__(self,v):
        self.value = v
    def accept(self,v): return v.visitIntLit(self)
class FloatLit(Exp):
    def __init__(self,v):
        self.value = v
    def accept(self,v): return v.visitFloatLit(self)
```

Let v1, v2, v3 be an object of Eval, PrintPrefix, Postfix and x be an object expressing an expression, x.accept(v1) will return the value of the expression x, x.accept(v2) will return the expression in prefix format and x.accept(v3) will return the expression in postfix format.

For testing, given some following objects:

```
x1 = IntLit(1)
x2 = FloatLit(2.0)
x3 = BinExp(x1,"+",x1)
x4 = UnExp("-",x1)
x5 = BinExp(x4,"+",BinExp(IntLit(4),"*",x2))
```

Be careful that you should not allowed to use type(), isinstance() when implementing this exercise.

Tip: Use Visitor pattern.

```
Reset answer
```

```
1 | class Eval: pass
2 | class PrintPrefix: pass
3 | class PrintPostfix: pass
4
```

