**Question 1.**

|  |  |
| --- | --- |
| *Grammar*  program  : ( expression )\* EOF  ;  expression  : Integer  | Identifier  ;  Integer: [0-9]+ ;  Identifier: [a-z]+ ; | *AST definition*  @data­­class  class Prog(AST):  expr : List[Exp]  class Exp(AST):  \_\_metaclass\_\_ = ABCMeta  pass  @dataclass  class Id(Exp):  def \_\_str\_\_(self):  return "ID"  @dataclass  class Int(Exp):  def \_\_str\_\_(self):  return "INT" |

Transformation table

|  |  |
| --- | --- |
| **Grammar** | **AST** |
| program | Prog |
| expression | Exp (abstract) |
| Integer | ID (Exp) |
| Identifier | Integer(Exp) |

Draw the parse trees and AST trees for the following strings

1. a
2. ab 12 ab 123

Solution:

a. Prog[ID(a)]

b. Prog[ID(ab), Integer(12), ID(ab), Integer(123)]

**Question 2.**

|  |  |
| --- | --- |
| *Grammar*  program  : ( expression )\* EOF  ;  expression  : expression '+' term  | term  ;  term  : Integer  | Identifier  ;  Integer: [0-9]+ ;  Identifier: [a-z]+ ; | *AST definition*  @dataclass  class Prog(AST):  expr : List[Exp]  class Exp(AST):  \_\_metaclass\_\_ = ABCMeta  pass  @dataclass  class Id(Exp):  def \_\_str\_\_(self):  return "ID"  @dataclass  class Int(Exp):  def \_\_str\_\_(self):  return "INT"  @dataclass  class BinOp(Exp):  op:str  left:Exp  right:Exp |

Transformation Table

|  |  |
| --- | --- |
| **Grammar** | **AST** |
| program | Prog |
| expression  term | Exp (abstract) |
| Integer | ID (Exp) |
| Identifier | Integer(Exp) |
|  | BinOp: string EXP EXP |

Draw the parse trees and AST trees for the following strings

1. a
2. a + b
3. a a + b

Solution:

a. Prog[ID(a)]

b. Prog[BinOp(ID(a),ID(b))]

c. Prog[ID(a), BinOp(a,b)]