1. Конкретный синтаксис.

$\langle ident \rangle ::= \langle nondigit \rangle$
  $| \quad \langle nondigit \rangle \ \langle alphanum \rangle$
  $| \quad \langle nondigit \rangle \ \langle alphanum \rangle \ \langle ident\_rest \rangle$

$\langle ident\_rest \rangle ::= \langle alphanum \rangle \ | \ \langle alphanum \rangle \ \langle ident\_rest \rangle$

$\langle cmp\text{-}op \rangle ::= \text{`=='} \ | \ \text{`/= '} \ | \ \text{`<='} \ | \ \text{`<'} \ | \ \text{`>'} \ | \ \text{`>='} \ \langle md\text{-}op \rangle ::= \text{`*'} \ | \ \text{`/'} \ \langle pm\text{-}op \rangle ::= \text{`+'} \ | \ \text{`.-'}$

$\langle unop \rangle ::= \text{'-'} \ | \ \text{'!'}$

$\langle num\text{-}lit \rangle ::= \text{number}$

$\langle bool\text{-}lit \rangle ::= \text{'T'} \ | \ \text{'F'}$

$\langle lit \rangle ::= \langle num\text{-}lit \rangle \ | \ \langle bool\text{-}lit \rangle$

$\langle expr \rangle ::= \langle binop\text{-}expr \rangle$

$\langle binop\text{-}expr \rangle ::= \langle or\text{-}expr \rangle$

$\langle or\text{-}expr \rangle ::= \langle and\text{-}expr \rangle \ \text{`||'} \ \langle or\text{-}expr \rangle \ | \ \langle and\text{-}expr \rangle$

$\langle and\text{-}expr \rangle ::= \langle cmp\text{-}expr \rangle \ \text{`\&\&'} \ \langle and\text{-}expr \rangle \ | \ \langle cmp\text{-}expr \rangle$

$\langle cmp\text{-}expr \rangle ::= \langle pm\text{-}expr \rangle \ \langle cmp\text{-}op \rangle \ \langle pm\text{-}expr \rangle \ | \ \langle pm\text{-}expr \rangle$

$\langle pm\text{-}expr \rangle ::= \langle pm\text{-}expr \rangle \ \langle pm\text{-}op \rangle \ \langle md\text{-}expr \rangle \ | \ \langle md\text{-}expr \rangle$

$\langle md\text{-}expr \rangle ::= \langle md\text{-}expr \rangle \ \langle md\text{-}op \rangle \ \langle pow\text{-}expr \rangle \ | \ \langle pow\text{-}expr \rangle$

$\langle pow\text{-}expr \rangle ::= \langle unop\text{-}expr \rangle \ \text{`\^'} \ \langle pow\text{-}expr \rangle \ | \ \langle unop\text{-}expr \rangle$

$\langle unop\text{-}expr \rangle ::= \langle un\text{-}op \rangle \ \langle lit \rangle$
  $| \quad \langle un\text{-}op \rangle \ \langle var \rangle$
  $| \quad \langle un\text{-}op \rangle \ \text{`('} \ \langle atom\text{-}expr \rangle \ \text{`)'}$

$\langle atom\text{-}expr \rangle ::= \langle ident \rangle$
  $| \quad \langle lit \rangle$
  $| \quad \langle app \rangle$
  $| \quad \text{'if'} \ \langle expr \rangle \ \text{'then'} \ \langle expr \rangle \ \text{'else'} \ \langle expr \rangle$
  $| \quad \text{'let'} \ \text{var} \ \text{'='} \ \langle expr \rangle \ \text{'in'} \ \langle expr \rangle$
  $| \quad \langle expr \rangle \ \langle binop \rangle \ \langle expr \rangle$
  $| \quad \langle unop \rangle \ \langle expr \rangle$
  $| \quad \text{'('} \ \langle expr \rangle \ \text{')'}$

$\langle app \rangle ::= \langle ident \rangle \ \text{' '} \ \langle ident \rangle$
  $| \quad \langle ident \rangle \ \text{' '} \ \langle lit \rangle$
  $| \quad \langle ident \rangle \ \text{' '} \ \text{'('} \ \langle expr \rangle \text{')'}$
  $| \quad \text{'('} \ \langle expr \rangle \ \text{')'} \ \langle ident \rangle$
  $| \quad \text{'('} \ \langle expr \rangle \ \text{')'} \ \langle lit \rangle$
  $| \quad \text{'('} \ \langle expr \rangle \ \text{')'} \ \text{'('} \ \langle expr \rangle \ \text{')'}$

$\langle bind \rangle ::= \langle ident \rangle \ \langle arg \rangle \ \text{'='} \ \langle expr \rangle$
  $| \quad \langle ident \rangle \ \text{'='} \ \langle expr \rangle$

$\langle arg \rangle ::= \langle ident \rangle \ | \ \langle ident \rangle \ \text{' '} \ \langle arg \rangle$

$\langle decl \rangle ::= \langle bind \rangle \ \text{';'} \ \langle decl \rangle \ | \ \langle bind \rangle \ \text{';'}$

Примеры.

1. Объявление функции/переменной.

   ```
   f x = x;
   n = 10;
   ```

2. Использование условного оператора if

   ```
   f x = if x == 0 then 10 else 20 + x;
   ```

3. Использование let-связывания.

   ```
   f y = let x = 10 * y in x ^ x;
   ```

4. Вызов функции

   ```
   f x = x;
   g = f 10;

   ff x y = x + y;
   gg = ff 10 20;
   ```

2. Абстрактный синтаксис

Представлен в виде АСД.

```
type Ast = [Decl]

data Decl = BindDecl Bind

data Bind = Bind Name [Name] Expr

data Expr = Var Name
          | Lit Lit
          | App Expr Expr
          | If Expr Expr Expr
          | Let Name Expr Expr
          | UnOp UnOperator Expr
          | BinOp BinOperator Expr Expr

data Lit = ILit Integer | BLit Bool
```