

# Bonk Language Grammar

Keywords: `flot`, `nubr`, `strg`, `many`

Operators: `@`, `+`, `-`, `*`, `/`, `+=`, `-=`, `*=`, `/=`, `=`, `==`, `<`, `>`, `<=`, `>=`, `!=`, `blok`, `help`, `brek`, `hive`, `bowl`, `bonk`, `loop`, `of`, `and`, `or`, `not`

Grammar:

*/\* Overall program structure \*/*

- Program : HelpStatement\* Declaration\*
- HelpStatement : `help` Identifier
- Declaration : BlokDeclaration | VariableDeclaration | HiveDeclaration
- BlokDeclaration : `blok` Identifier ParameterListDeclaration? CodeBlock
- VariableDeclaration : `bowl` Identifier (: Type)? (= Expression)?
- ParameterListDeclaration: [(VariableDeclaration (, VariableDeclaration)\*)?]
- CodeBlock: { (CodeBlock | LoopStatement | Statement;)\* }
- Statement: { Expression | BonkStatement | BrekStatement | VariableDeclaration }
- BonkStatement: `bonk` Expression?
- BrekStatement: `brek`
- ArrayConstant: [Expression\*]
- HiveAccess: Identifier `of` ExpressionPrimary
- ParameterList: [(ParameterListItem (, ParameterListItem)\*)?]
- ParameterListItem: Identifier = Expression
- LoopStatement: `loop` ParameterListDeclaration? CodeBlock
- Type: `many` Type | TrivialType | Identifier
- HiveDeclaration: `hive` Identifier { (BlokDeclaration | VariableDeclaration;)\* }

*/\* All kinds of expressions, operator precedence \*/*

- **OperatorExpression**<NextExpression, Operator>:
  - NextExpression |
  - NextExpression Operator OperatorExpression<NextExpression, Operator>
- Expression: ExpressionAssignment
- ExpressionAssignment: **OperatorExpression**<ExpressionOr, (= | += | -= | \*= | /=)>
- ExpressionOr: **OperatorExpression**<ExpressionAnd, `or`>
- ExpressionAnd: **OperatorExpression**<ExpressionEquality, `and`>
- ExpressionEquality: **OperatorExpression**<ExpressionRelational, (== | !=)>
- ExpressionRelational: **OperatorExpression**<ExpressionAdd, (< | > | <= | >=)>
- ExpressionAdd: **OperatorExpression**<ExpressionMul, (+ | -)>
- ExpressionMul: **OperatorExpression**<ExpressionUnary, (\* | /)>
- ExpressionUnary:
  - ExpressionPrimary |
  - ExpressionCall |
  - `UnaryOperator` ExpressionUnary
- ExpressionPrimary: HiveAccess | Identifier | NumberConstant | StringConstant | ArrayConstant | (Expression) | CodeBlock
- ExpressionCall: @ Expression ArgumentList?

# Example program

```
hive Persn {
    bowl hapiness: flot;

    blok hapy_birthday[bowl how_hapy: nubr] {

        hapiness of me += how_hapy;

        bowl messag: strg;

        dogo: There are no if-s, instead one should
        dogo: use lazy expression calculation, like so:

        how_hapy > 0
            and messag = "I am so happy it's my birthday!"
            or messag = "I'm getting closer and closer to death :c";

        bowl messages: many strg = ["It's my birthday!"];

        loop[bowl counter = 0] {
            counter += 1 < how_hapy of me or brek;
            messages += text;
        }

        loop[bowl counter = 0] {
            counter += 1 < length of messages or brek;

            @print[text = @value of messages[index = counter]];
        }
    }
}

blok main {
    bowl you = @Persn[hapiness = 5];
    bowl he = @Persn[hapiness = 6];

    @hapy_birthday of you[how_hapy = 10];

    hapiness of he = hapiness of you;

    bonk hapiness of you;
}
```