# Parthonyte Grammar

[ Go Back ]

At least one occurrence of a white space character (or a comment block), open parenthesis, close parenthesis, or semicolon occurs between adjacent tokens. A comment block consists of a pair of brace brackets enclosing zero or more characters.

## Grammar Notation

- Non-terminal symbol: <symbol>
- Optional text in brackets: [ *text* ]
- Repeats zero or more times: [ *text* ]…
- Repeats one or more times: <symbol>…
- Pipe separates alternatives: *opt1* | *opt2*
- Comments in *italics*

<source file>:
- do ( [<imp>]... [<def glb>] [<def>]... [<class>]... )

<imp>:
<import stmt> **;**

<import stmt>:
import <module>...
from <rel module> import <mod list>
from <rel module> import all

<module>:
<name>
( **:** <name><name>... )
( as <name><name> )
( as ( **:** <name><name>... ) <name> )

<mod list>:
<id as>...

<id as>:
<mod id>
( as <mod id><name> )

<mod id>:
<mod name>
<class name>
<func name>
<var name>

<rel module>:
( **:** [<num>] [<name>]... )
<name> *// ?*

<cls typ>:
class
iclass

<hedron>:
hedron
ihedron

<class>:
- <cls typ><name> [<base class>] [<does>] [<gvars>] [<ivars>] do ( <def>… ) **;**
- abclass <name> [<base class>] [<does>] [<gvars>] [<ivars>] do ( <anydef>… ) **;**
- <hedron><name> [<does>] [<const list>] do ( [<abdef>]... [<defimp>]... ) **;**
- enum <name><elist> **;**
- ienum <name><elist> **;**

<does>:
( does <hedron name>... )

<hedron name>:
<base class>:
<name>
( **:** <name><name>… )

<const list>:
( const <const pair>... )

<const pair>:
( <name><const expr> )

<def glb>:
- gdefun [<vars>] [<gvars>] [<ivars>] do <block> ;

<def>:
- <defun> ( <name> [<parms>] ) [<vars>] [<gvars>] [<dec>] do <block> **;**

<defimp>:
- defimp ( <name> [<parms>] ) [<vars>] [<gvars>] [<dec>] do <block> **;**

<abdef>:
abdefun ( <name> [<parms>] ) [<dec>] **;**

<defun>:
defun
idefun

<anydef>:
    <def> | <abdef>

<vars>:
    ( var [<id>]... )

<ivars>:
    ( ivar [<id>]... )

<gvars>:
    *// added to class/gdefun doc: Nov/24*
    ( gvar [<id>]... )

<parms>:
    [<id>]... [<parm>]... [ ( * <id> ) ] [ ( ** <id> ) ]

<parm>:
    ( <set op><id><const expr> )

<dec>:
    ( decor <dec expr>... )

<block>:
    ( [<stmt-semi>]… )

<stmt-semi>:
    <stmt> **;**

<jump stmt>:
    <continue stmt>
    <break stmt>
    <return stmt>
    return <expr>
    <raise stmt>

<raise stmt>:
    raise [<expr> [ from <expr>] ]

<stmt>:
    <if stmt>
    <while stmt>
    <for stmt>
    <switch stmt>
    <try stmt>
    <asst stmt>
    <del stmt>
    <jump stmt>
    <call stmt>
    <print stmt>
    <bool stmt>

<call expr>:
- ( <name> [<arg list>] )
- ( **:** <colon expr>… <name> )
- ( **:** <colon expr>… ( <method name> [<arg list>] ))
- ( **::** <colon expr>… <name> else <expr> )
- ( **::** <colon expr>… ( <method name> [<arg list>] ) else <expr> )
- ( call <expr> [<arg list>] )

<call stmt>:
- <name> [<arg list>]
- **:** <colon expr>… ( <method name> [<arg list>] )
- call <expr> [<arg list>]

<colon expr>:
    <name>
    ( <name> [<arg list>] )

<arg list>:
    [<expr>]... [ ( <set op><id><expr> ) ]...

<dec expr>:
    <name>
    ( <name><id>... )
    ( **:** <name><id>... )
    ( **:** <name>... ( <id>... ))

<dot op>:
    dot | **:**

<dotnull op>:
    dotnull | **::**

<del stmt>:
    del <expr>

<set op>:
    set | =

<asst stmt>:
    <asst op><target expr><expr>
    <set op> ( tuple <target expr>... ) <expr>
    <inc op><name>

<asst op>:
    set | addset | minusset | mpyset | divset |
    idivset | modset |
    shlset | shrset | shruset |
    andbset | xorbset | orbset |
    andset | xorset | orset |
    = | += | -= | *= | /= |
    //= | %= |
    <<= | >>= | >>>= |
    &= | ^= | '|=' |
    &&= | ^^= | '||='

<target expr>:
    <name>
    ( **:** <colon expr>… <name> )
    ( slice <arr><expr> [<expr>] )
    ( slice <arr><expr> all )
    ( <crop><cons expr> )

<arr>:  *// string or array/list*
    <name>
    <expr>

<if stmt>:
- if <expr> do <block> [ elif <expr> do <block>]…
  [ else do <block>]

<while stmt>:
    while <expr> do <block>
    while do <block> until <expr>

<for stmt>:
- for <name> [<idx var>] in <expr> do <block>
- for ( <bool stmt>**;** <bool stmt>**;** < bool stmt> )
  do <block>

<try stmt>:
- try do <block> <except clause>… [ else do
  <block>] [ eotry do <block>]
- try do <block> eotry do <block>

<except clause>:
    except <name> [ as <name>] do <block>

<bool stmt>:
    quest [<expr>]
    ? [<expr>]
    <asst stmt>

<switch stmt>:
    switch <expr><case body> [ else do <block>]

<case body>:
    [ case <id> do <block>]...
    [ case <dec int> do <block>]...
    [ case <str lit> do <block>]...
    [ case <tuple expr> do <block>]...

<swix expr>:
    ( swix <expr><swix body> [ else <expr>] )

<swix body>:
    [ ( case <id><expr> ) ]...
    [ ( case <dec int><expr> ) ]...
    [ ( case <str lit><expr> ) ]...
    [ ( case <tuple expr><expr> ) ]...

<return stmt>:
    return

<break stmt>:
    break

<continue stmt>:
    continue

<paren stmt>:
    ( <stmt> )

<qblock>:
    ( quote [<paren stmt>]... )

<quest>:
    quest | ?

<cquest>:
    cquest | ??

<inc op>:
    incint | decint | ++ | --

<expr>:
    <keyword const>
    <literal>
    <name>
    ( <unary op><expr> )
    ( <bin op><expr><expr> )
    ( <multi op><expr><expr>… )
    ( <quest><expr><expr><expr> )
    ( <cquest> [ ( case <expr><expr> ) ]... )
    <swix expr>
    <lambda>
    ( quote <expr>... )
    <cons expr>
    <tuple expr>
    <list expr>
    <dict expr>
    <venum expr>
    <string expr>
    <bytes expr>
    <target expr>
    <call expr>
    <cast>

<unary op>:
    minus | notbitz | not |
    - | ~ | !

<bin op>:
    <arith op>
    <comparison op>
    <shift op>
    <bitwise op>
    <boolean op>

<arith op>:
    div | idiv | mod | mpy | add | minus |
    / | // | % | * | + | -

<comparison op>:
    ge | le | gt | lt | eq | ne | is | in |
    >= | <= | > | < | == | !=

<shift op>:
    shl | shr | shru |
    << | >> | >>>

*Note: some operators delimited with single quotes for clarity (quotes omitted in source code)*

<bitwise op>:
    andbitz | xorbitz | orbitz |
    & | ^ | '|'

<boolean op>:
    and | xor | or |
    && | ^^ | '||'

<multi op>:
    mpy | add | strdo | strcat |
    and | xor | andbitz | xorbitz |
    or | orbitz |
    * | + | % | + |
    && | ^^ | & | ^ |
    '||' | '|'

<const expr>:
    <literal>
    <keyword const>

<literal>:
    <num lit>
    <str lit>
    <bytes lit>

<cons expr>:
    ( cons <expr><expr> )
    ( <crop><expr> )

<tuple expr>:
    ( tuple [<expr>]… )
    ( <literal> [<expr>]… )
    ( )

<list expr>:
    ( lyst [<expr>]… )

<dict expr>:
    ( dict [<pair>]… )

<pair>:
    // expr1 is a string
    ( **:** <expr1><expr2> )
    ( **:** <str lit><expr> )

<venum expr>:
    ( venum <enum name> [<elist>] )
    ( venum <enum name><idpair>... )

<elist>:
    <id>...
    <intpair>...
    <chpair>...

<intpair>
    // integer constant
    <int const>
    ( **:** <int const><int const> )

<chpair>
    // one-char. string
    <char lit>
    ( **:** <char lit><char lit> )

<idpair>
    <id>
    ( **:** <id><id> )

<cast>:
    ( cast <literal><expr> )
    ( cast <class name><expr> )

<print stmt>:  *// built-in func*
    print <expr>…
    println [<expr>]…
    echo <expr>…

<lambda>:
    ( lambda ( [<id>]... ) <expr> )
    ( lambda ( [<id>]... ) do <block> )
    ( lambdaq ( [<id>]... ) do <qblock> )
    *// must pass qblock thru compile func*

*No white space allowed between tokens, for rest of Parthonyte Grammar*

<white space>:
    <white token>...

<white token>:
    <white char>
    <line-comment>
    <blk-comment>

<line-comment>:
    # [<char>]... <new-line>

<blk-comment>:
    {# [<char>]... #}

<white char>:
    <space> | <tab> | <new-line>

<name>:
- [<underscore>]… <letter> [<alnum>]… [<hyphen-alnum>]… [<underscore>]… [<alnum>]…

<hyphen-alnum>:
    <hyphen><alnum>…

<alnum>:
    <letter>
    <digit>

*In plain English, names begin and end with zero or more underscores (followed by optional alphanumeric characters). In between is a letter followed by zero or more alphanumeric characters. Names may also contain hyphens, where each hyphen is preceded and succeeded by an alphanumeric character.* ***Optional alnum* suffix added 24-Nov-24***

<num lit>:
    <dec int>
    <long int>
    <oct int>
    <hex int>
    <bin int>
    <float>

<dec int>:
    [<hyphen>] 0
    [<hyphen>] <any digit except 0> [<digit>]…

<long int>:
    <dec int> L

<float>:
    <dec int><fraction> [<exponent>]
    <dec int><exponent>

<fraction>:
    <dot> [<digit>]…

<exponent>:
    <e> [<sign>] <digit>…

<e>:
    e | E

<sign>:
    + | -

<keyword const>:
    null
    true
    false

<oct int>:
    0o <octal digit>…

<hex int>:
    0x <hex digit>…
    0X <hex digit>…

<bin int>:
    0b <zero or one>…
    0B <zero or one>…

<octal digit>:
    0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

<hex digit>:
    <digit>
    A | B | C | D | E | F
    a | b | c | d | e | f

<str lit>:
    " [<str item>]... "

<str item>:
    <str char>
    <escaped str char>
    <str newline>

<str char>:
    any source char. except "\", newline, or end quote

<str newline>:
    \ <newline> [<white space>] "

<escaped char>:
    \\  *backslash*
    \"  *double quote*
    \a  *bell*
    \b  *backspace*
    \f  *formfeed*
    \n  *new line*
    \r  *carriage return*
    \t  *tab*
    \v  *vertical tab*
    \ooo  *octal value = ooo*
    \xhh  *hex value = hh*

<escaped str char>:
    <escaped char>
    \N{name}  *Unicode char. = name*
    \uxxxx  *hex value (16-bit) = xxxx*

<crop>:
    c <crmid>... r

<crmid>:
    a | d

*Not implemented: string prefix and bytes data type (rest of grammar)*

<str lit>:
    [ $ <str prefix>] <quoted str>

<str prefix>:
    r | R

<quoted str>:
    " [<str item>]... "

<bytes lit>:
    $ <byte prefix><quoted bytes>

<byte prefix>:  *// any case/order*
    b | br

<quoted bytes>:
    " [<bytes item>]... "

<bytes item>:
    <bytes char>
    <escaped char>
    <str newline>

<bytes char>:
    any ASCII char. except "\", newline, or
    end quote