

# Python

- with type hints
- without round brackets in the while, for, function
- with brackets after while, for, function
- variables must start with '\_'

## Lexic.txt:

Alphabet :

- upper and lower case letters of the English alphabet
- Underline character '\_';
- Decimal digits (0-9)

Identifiers :

- an '\_' followed by letters or digits

Constants :

- int
  - non\_zero\_digit = 1 | .. | 9
  - digit = 0 | non\_zero\_digit
  - unsigned\_int = non\_zero\_digit {digit}
  - signed\_int = ['+' | '-'] unsigned\_int
- char
  - letter = 'a' | ... | 'z' | 'A' | ... | 'Z'
  - char = 'letter' | 'digit'
- string
  - string = char | {char}

Special symbols :

- arithmetic operators: + - \* / %
- relational operators: = != < <= == => > ?
- separators: {} [] () : ; space ?
- reserved words:
  - int char string vector for if while for read print

## token.in:

```

[reserved_words]
int
string
vector
if
else
while
for
read
print
[operators]
+
-
*
/
%
=
==
!=
<
<=
>=
||
&&
[separators]
[
]
{
}
;
,
'
"

```

## Syntax.in:

```

- <program> ::= null | <statement-list>
- <statement-list> ::= <statement> | <statement-list> <statement>
- <statement> ::= <declaration statement> | <assignment statement> | <io statement> | <if statement> | <while s
- <for statement> ::= for <statement>, <condition>, <statement> { <statement-list> }
- <relational operator> ::= = | < | <= | == | != | > | >
- <while statement> ::= while <condition> { <statement-list> }
- <if statement> ::= if <condition> { <statement-list> } | if <condition> {<statement-list>} else {<statement-
- <io statement> ::= read(<identifier>) | print(<identifier>)
- <assignment statement> ::= <identifier> = <expression>
- <vector declaration> ::= vector(<type>) <identifier>
- <declaration statement> ::= <type> <identifier> | <type> <identifier> = <expression>
- <condition> ::= <expression> <relational operator> <expression>
- <expression> ::= <expression> <expression operator> <term> | <term>
- <expression operator> ::= + | -
- <term> ::= <term> <term operator> <factor> | <factor>
- <term operator> ::= * | / | %
- <factor> ::= (<expression>) | <identifier> | <constant>
- <identifier> ::= _ | <identifier> <letter> | <identifier> <digit>
- <digit> = 0 | ... | 9

```

```
- <letter> = a | ... | z | A | ... | Z
- <type> ::= int | char | string | vector
```

## Complex Types:

`vector(basic_type)` - stores lists of values

## Variables

```
int _a = 5
string _c = "Hello"
vector(int) _l
```

## function

```
| function sum int _a, int _b {
|     return _a + _b
| }
```

## while

```
int _a = 0
while _a < 10 {
    a = a + 1
}
```

## for

```
for _i = 0, _i < 10, _i = _i + 1 {
    print(i)
}
```

## if

```
int _a = 5
if _a > 0 {
    print("Positive")
}
else {
    print("Negative")
}
```

## I/O

```
read(variable) # reading
print(variable) # printing
```

### p1.vladpy

greatest common divisor (GCD) of two numbers

```
int _a, _b
read(_a)
read(_b)
while _b != 0 {
    _copy_b = _b
    _b = _a % _b
    _a = _b
}
print(_a)
```

### p2.vladpy

factorial of a number

```
int _n
read(_n)
int _fact = 1
for _i = 1, _i < _n + 1, _i = _i + 1 {
    _fact = _fact * _i
}
print(_fact)
```

### p3.vladpy

function that does the binary search

```
vector(int) _l = [1, 2, 3, 4, 5]

function binarySearch vector(int) _l, int _x {
    int _left = 0
    int _right = len(_l) - 1
    while _left <= _right {
        int _mid = _left + (_right - _left) / 2
        if _l[_mid] == _x {
            return _mid
        }
        if _l[_mid] < _x {
            _left = _mid + 1
        }
        else {
            _right = _mid - 1
        }
    }
}
```

```
        }
    }
    return -1
}
```

## pError.vladpy

```
_n # _n does not have type declaration
read(_n)
int _fact = 1
for _i in range(1, _n+1) # the brackets are missing
    _fact = _fact * _i
print(_fact)
```