

EvalPrint(EP) Grammar

```
<program>                :: (<binding> | <expr> | <func-def>)* EOF
<binding>                 :: let := <binding-name>
<expr>                    :: <expr> <bin_op> <expr>
                           | (<expr>)
                           | <func-call>
                           | <binding-name>
                           | <number>
                           | <unary-op> <expr>
<func-def>                | func <binding-name>(<binding-name>*) := <expr>
<bin_op>                  | +
                           | *
                           | ^
                           | -
                           | /
                           | //
                           | %
<unary-op>                | +
                           | -
<func-call>               | <binding-name>(<binding-name>*)
<number>                  :: <real-number>
                           | <complex-number>
<real-number>             :: <float-number>
                           :: <int-number>
<signed-real-number>      :: [+]<real-number>
                           | [-]<real-number>
<complex-number>          :: complex(<signed-real-number>, <signed-real-number>)
<int-number>              :: <hex-number>
                           | <bin-number>
                           | <dec-number>
                           | <oct-number>
<float-number>            :: Any float number e.g 10.784, 1e10, .145
<hex-number>              :: Any hex    number e.g 0x00FF
<bin-number>              :: Any bin    number e.g 0b0013
<oct-number>              :: Any oct    number e.g 0o1453
<dec-number>              :: Any decimal number e.g 1, 2, 10
<binding-name>            :: Any string which starts with an alphabetic letter or
                           "_" followed by alphanumeric characters or "_" e.g add, add_1, add_, _
```

Notes

- `//` is integer division, e.g
- Terminal symbols are in bold
- `\n` indicates we should move to a new line
- Splitting statements with multiple newlines will cause a problem
- We can call all python functions in the match library
- When calling functions with complex arguments, we use prepend the function name with `c_`. For instance instead of `cos(complex(1, 2))`, we call

`c_cos(complex(1, 2))`. Instead of `tan(complex(1, 2))` we call `c_tan(complex(1, 2))`

Examples of programs that would be accepted by the tokenizer are:

1:

```
1 * 2
```

2:

```
(-1)
```

3:

```
18494.784 ^ 3
```

4:

```
(1 * 48 * a * 389 * 64)
```

5:

```
func multiply(a, b) := a * b  
multiply(a, b)
```

6:

```
let a := 10  
a * 10
```

7:

```
let a := 10.3  
let c := 15.4  
1944.66 + c * 3.4 + 1.4 + 2.3 + 4.5  
(-2.0) + a  
let b := 10.2  
a + b  
let d := 10  
func add(x, y) := x + y  
add(10, 20)  
d + 30
```

8:

```
c_cos(complex(1, 2))
```