Лексический синтаксис

1. Идентификатор **<id>** (name) – непустая последовательность букв латинского алфавита в любом регистре, цифр и символа нижнего подчеркивания (_), начинающаяся на букву латинского алфавита в нижнем регистре, не являющаяся ключевым словом.

   • Корректные идентификаторы: x, list, listNat_123.

   • Некорректные идентификаторы: Abc, 123, _List.

2. Число **<num>**: натуральное или ноль в десятичной системе счисления, не может содержать лидирующие нули.

   • Корректные числа: 123, 0.

   • Некорректные числа: -1, 007, 89A.

3. Ключевые слова **<keyword>** (не могут быть идентификаторами):
   - def – инструкция для определения функции;
   - while – инструкция для определения цикла с предусловием
   - if – условный оператор ветвления, реализует выполнение определённых команд при условии, что некоторое логическое выражение принимает значение true
   - else – реализует выполнение определённых команд при условии, что выражение в “if” принимает значение false
   - return – возвращение значения из функции
4. Операторы языка **<operator>**:
   - сложение +,
   - умножение *,
   - деление /,
   - вычитание -,
   - возведение в степень **,
   - конъюнкция &&,

- дизъюнкция ||,
- логическое отрицание --,
- операторы сравнения: <=, ==, /=, >, >=

Пробелы не являются значимыми, но не могут встречаться внутри одной лексемы.

Конкретный синтаксис

1. Программа — непустая последовательность определений функций.
2. Определение функции содержит ее сигнатуру и тело. Сигнатура функции содержит ее название (идентификатор) и список аргументов (может быть пустым). Тело — последовательность инструкций (может быть пустой):
   def <id> (<id>, <id>) {
   <body>
   }
   Например,
   def mult() {
   }

   def mult(a, b) {
   c = a * b;
   }

   def mult(a, b, c, d) {
   e = a * b * c * d;
   return c;
   }

3. Инструкции

3.1. Присвоение значения арифметического выражения переменной. Переменная может быть произвольным идентификатором:

a = <num>;

b = <num>;

sum = a + b;

Например,

a = 1;

b = 2;

sum = a + b;

mult = a * b;

pow = a ** b;

3.2. Возвращение значения из функции:

Например,

def sum(a, b, c, d) {

e = a + b + c + d;

return e;

}

3.3. Условное выражение с обязательной веткой else. Условием является арифметическое выражение. В ветках — произвольные последовательности инструкций (могут быть пустыми):

if (<id>==<id> && <id>==<num>) {

return <id> + <num>;

}

else {

return <id>;

}

Например,

if (a == b || c = 2) {

return b ** c + 1;

}

else {

return b + c;

}

3.4. Цикл с предусловием. Условием является арифметическое выражение. Тело цикла — произвольная последовательность инструкций (может быть пустой):

while (<id> >= <num>) {

<id> = <id> - <num>;

}

Например,

while (number >= 0) {

number = number - 2;

}

4. Программа – непустая последовательность определений функций.

Например,

```
def pow(a, b) {
res = a ** b;
return res;
}
def main() {
a = 3
b = 4
result = pow(a, b)
}
```

5. В языке бывают вызовы функций. Например, f(x,y) + g(y). Они могут встречать также, как переменные и числа, а также быть инструкциями.

```
def pow(x, y) {
result = x ** y;
return result;
}
```

```
def g(y) {
sum = y + 10;
return sum;
}
def main () {
a = 10;
b = 2;
c = 5;
res = pow(a, b) + g(c);
}
```

6. Арифметические выражения заданы над числами и идентификаторами, операторы перечислены в таблице ниже с указанием их приоритетов, арности и ассоциативности.

Наибольший приоритет	Арность	Ассоциативность
-	Унарная	
**	Бинарная	Правоассоциативная
*,/	Бинарная	Левоассоциативная
+,-	Бинарная	Левоассоциативная
==,/=, <,<=, >,>=	Бинарная	Неассоциативная
--	Унарная	
&&	Бинарная	Правоассоциативная
||	Бинарная	Правоассоциативная
Наименьший приоритет	Арность	Ассоциативность