BNF

Syntax, grammar, BNF

Syntax

The **syntax** of a language is concerned with

- which characters are allowed to appear in the source code
- and in which order

regardless of their meaning (“semantics”)

Examples of syntaxically invalid Python code:

```
print False
```

```
$4 + ¢20
```

```
log(😅) == 💧log(😄)
```

Examples of syntaxically valid (but semantically invalid) Python code:

```python
print(this_variable_was_never_created)
```

```python
a = 3
b = "2"
a + b
```

```python
a = 3
b = 2
print(str(a) + ' + ' + str(b) + ' = ' + str(a ** b))
```

Grammar and BNF

The **grammar** of a language is a set of rules that formally define valid syntax.

The Backus–Naur Form (**BNF**) is a notation to formalize this grammar.

It takes the general form

```
<id> ::= sequence | sequence | ... | sequence
```

where `sequence` is a sequence of “terminals” (i.e. raw strings) and other `<id>`s

Examples

Digits

```
<dec-digit> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

```
<hex-digit> ::= <dec-digit> | "a" | "b" | "c" | "d" | "e" | "f"
```

Numbers

```
<dec-digit> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

```
<dec-1-or-2-digits> ::= <dec-digit> | <dec-digit> <dec-digit>
```

```
<dec-1-or-2-or-3-digits> ::= <dec-digit> | <dec-digit> <dec-digit> | <dec-digit> <dec-digit> <dec-digit>
```

```
<dec-1-or-2-or-3-digits> ::= <dec-digit> | <dec-digit> <dec-1-or-2-digits>
```

```
<dec-1-or-2-or-3-or-4-digits> ::= <dec-digit> | <dec-digit> <dec-1-or-2-or-3-digits>
```

```
<dec-1-or-2-or-3-or-4-or-5-digits> ::= <dec-digit> | <dec-digit> <dec-1-or-2-or-3-or-4-digits>
```

```
<dec-number> ::= <dec-digit> | <dec-number> <dec-digit>
```

Numbers

```
<dec-digit> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
<hex-digit> ::= <dec-digit> | "a" | "b" | "c" | "d" | "e" | "f"

<dec-number> ::= <dec-digit> | <dec-number> <dec-digit>
```

```
<hex-seq> ::= <hex-digit> | <hex-seq> <hex-digit>
```

```
<number> := <dec-number> | "0x" <hex-seq>
```

Complex recursion

```
<paren-expr> ::= <number> | "(" <expr> ")"
<expr> ::= <paren-expr> | <paren-expr> "+" <paren-expr>
```

Valid `<expr>`:

```
<number>
```

```
( <expr> )
```

```
<number> + <number>
<number> + ( <expr> )
( <expr> ) + <number>
( <expr> ) + ( <expr> )
```

```
<number> + <number>
<number> + ( <number> )
<number> + ( ( <expr> ) )
<number> + ( <number> + <number> )
<number> + ( <number> + ( <expr> ) )
<number> + ( ( <expr> ) + <number> )
<number> + ( ( <expr> ) + ( <expr> ) )
...
```